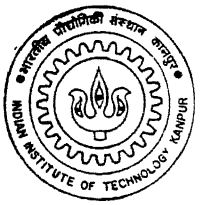
AN APPROACH TO UNIVERSITY COURSE TIMETABLING USING GENETIC ALGORITHMS

by

ARVIND K. TRIPATHI



DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1997

AN APPROACH TO UNIVERSITY COURSE TIMETABLING USING GENETIC ALGORITHMS

A thesis submitted

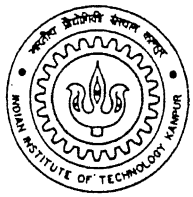
in partial fulfillment of the requirements

for the degree of

MASTER OF TECHNOLOGY

by

Arvind K Tripathi



to the

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 1997

ME-1997-M-TRI-APP

CERTIFICATE

This is to certify that the work contained in the thesis entitled *An Approach to University Course Timetabling using Genetic Algorithms* by Arvind K Tripathi, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Dr. Kalyanmoy Deb
Asst. Prof.
Department of Mech. Engg.
Indian Institute of Technology,
Kanpur.

Dr. T. P. Bagchi
Professor
Department of IME
Indian Institute of Technology,
Kanpur.

April 1997

Acknowledgements

I take this oportunity to express my sense of gratitude to my supervisors, Dr. Kalyanmoy Deb and Dr. T. P. Bagchi for their warm guidance that led me to the completion of this work. I express my sincere thanks to Dr. Kalyanmoy Deb who has introduced me to the field of GAs and motivated me to work in this emerging and challenging area of research. I am thankful to Prof. S. G. Dhande for extending the facilities of the CAD lab for completion of this work. Also I would like to thank Mr. Neeraj for the valuable discussions related to this work.

My wife, also my colleague, has an unshakable faith in me and she readily concurs all my decisions. She is ready to undego any amount o f sufferings for my benefit. Her constant support and encouragement during the course of this program was the source of energy to achieve my goal. She never complained, though she had to undergo great amount of sufferings being away from me during the course of this program.

I express my deepest gratitude to my parents for their invaluable love

and affection. Thanks to my parents, Bhaiya and Bhabhi for their constant inspiration in my effort to do this work.

I would like to thank my friends Atul, Vikas, Kumarswami, Bipin, Makrand, Vermaji and Manoj Kumar for their warm affections to make my stay at IIT Kanpur a pleasant one. Also I thank to Sh. A. D. Bhatt and all the CAD lab colleagues and staff for their kind support.

Abstract

Finding an adequate schedule of classes by properly utilizing resourses such as classrooms, working hours and instructors is a daunting task in any university. The problem, by no means is small for a smaller university, simply because although a fewer courses are offered, the available resourses in these universities are also less. The goal of a university course timetabling task is to find a feasible schedule which does not have conflicts in assigning courses for instructors, in rooms and in class hours.

In this thesis, three different techniques are developed and compared with each other to find the efficient one for feasible course timetabling. First, a simple Heuristic-based scheduling algorithm is developed based on sequential assignments. Second, the course timetabling problem is formulated with standard Linear Integer Programming method and solved using the LP software "CPLEX". Third, a Genetic Algorithm(GA) formulation is attempted. In a systematic study for solving course timetabling problem with varying degrees of difficulties(5 to 80 courses), it is observed that GA can find a fea-

sible schedule for all problems, whereas other two methods are not successful in more complex problems either due to lack of efficient search capabilities(in Heuristic search) or due to lack of adequate computational resourses(in LIP method). The results suggest that formulation proposed in this thesis is efficient and GA is an ideal tool to solve the complex course scheduling problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Scheduling is the allocation of resourses over time to perform a collection of tasks"[Bak74]. Course scheduling is a familiar and difficult scheduling problem where tasks to be scheduled are meetings. Resourses include instructors, classrooms etc. Effective course scheduling maximizes the likelihood that students can get desired courses, while considering other goals and constraints [Wer85].

The timetabling problem has been of interest since late 1950's. Various researchers have dealt with the problem in different ways. The problem is of interest even now, largely due to its varied nature, complexity and last but not the least, its large size. A typical data from Purdue University shows that in each Fall semester, approximately 35000 students pursue

degrees in 40 different curricula, enroll in over 3000 courses offered by 80 academics departments. These courses are taught in roughly 8000 sections by approximately 2400 faculty members. Sections meet in classes having 1 to 500 students typically for three hours of lecture per week in available 268 classrooms[ELMP96].

Since most universities use a timetable for scheduling courses, a feasible solution to such complicated problems often exists. The current approach in many universities all over the globe, typically involve a heuristic assignment algorithm with a large manual input if automation is used at all. The problem that appears in all complex timetabling problems is the variety and large number of, sometimes non-compatible constraints. Heuristic assignment algorithms do not easily allow for the possibility of search and strongly limit the type of constraints that may be catered for.

Scheduling is an active area of research in Operations Research. Scheduling problems typically comprise several concurrent (and often conflicting) goals and several resourses which may be allocated in order to satisfy these goals[JAJK96]. In many cases the combination of goals and resourses results in an exponentially growing problem space. As an immediate result, no standard analytical methods exist for solving these problems in polynomial time. Such problems are called NP-complete problems, with respect to the exponential time and memory requirements necessary to reach optimal solution. Since scheduling problems challenge existing search methodologies, and since

successful solution of real world scheduling problems is an important focus in research, it has been a favourite domain among researchers.

Traditional scheduling heuristics can be verified by testing the method against the certain benchmarks within a particular problem space or by comparing the solution with a lower bound. Both these approaches have shortcomings. As it is not possible to test a heuristic exhaustively in any practical application domain [Deb95], there is always a possibility that certain untested solution in the solution space can lead to poor performance of the algorithm. It is also at times difficult to determine a meaningful lower bound to a problem[DD96].

Genetic Algorithms were developed in mid-1960s by John Holland and further a lot of work has been done to bring it to its present status by David Goldberg and other researchers. GAs seek to breed good solutions to complex problems by a paradigm that mimics natural evolution (survival of fittest)[Gol89a]. One property of GA which makes it an attractive tool in the domain of scheduling is its flexibility as it is easy to incorporate the problem specific peculiarities in the form of penalties to the solution[ST93].

First it arrives at a result by improving on the best solution over a number of iterations(generations). The best solution from each previous generation provide at each stage a history of the performance of GA. This history gives the user insight into the behaviour of the GA on each problem

Second, a GA has certain input parameters such as the initial population size [Gol89b], the crossover and mutation rate which can be varied to try to improve on poor performance[Mic92]. The aim of the present work is to develop an efficient technique to deal with similar kind of the problem as discussed. For this we have organised thesis in the following manner:

In Chapter 2, first we have discussed the problem and some of the constraints we are considering, then we discuss the previous work which has been done in this field. Next section of the same Chapter discusses the formulation of the problem. In formulation we have discussed first the Heuristic Search Method with all the constraints and algorithm followed. Further we discussed the formulation by Linear Integer Programming Method with same constraints we kept in Heuristic search method. And lastly we discussed our proposed GA Approach and chromosome representation.

In Chapter 3, we have discussed the results of three test problems for all three methods to attempt a comparison of the efficiency and reach of all three methods. Also, in the same Chapter the factorial design[BDss] for GA approach has been attempted. Chapter 4 discusses the conclusions and Extensions based on results.

Chapter 2

Problem Formulation and Modeling

2.1 Problem Statement

The poor utilization of available classrooms and the dissatisfaction of faculty with their course schedules are common problems encountered when attempting to schedule university classes.

Facing with increasing students number, with new courses introduced, with shortage of lecture rooms and laboratories, and with growing number of courses open for students of departments, hence with a large number of conflicting constraints, timetables which can be largely accepted by faculty

and students are very difficult to schedule in most of the universities[FR85]. From the faculty's point of view, the objective is to maximize their preferences of courses and other requirements, if any; from the administration's point of view, the effective utilization of physical facilities is a concern as well.

The problem of University course scheduling, we have formulated, with usual constraints is[Tri84]:

- All the lectures of all courses must be scheduled.
- Limit on maximum number of assignments for instructors per week, i.e. one instructor can take maximum 6 lectures per week.
- Preferences of instructors for courses to be taken.
- Specific classroom requirement of particular course. In our problem, for every course, only two options are given for room and two options for instructor.
- Any instructor will not take consecutive classes on any day.
- Any instructor will not be assigned more than one lecture at a time.
- A room can not be assigned more than one lecture per time period.
- Two schemes for allocating lectures of any course were followed, either

scheme 1(Monday, Wednesday, Friday) or scheme 2(Tuesday, Thursday, Friday/Monday.

We have considered a few constraints in formulation of our test problems, but there could be many more in real university timetabling problem. Also the constraints governing the scheduling of courses largly depend on specific requirements of universities. The consideration of course assignments are important preferences for instructors. The instructors may express preferences for certain times of a day, certain days of a week etc., in addition, because of pressures on room utilization, there may be a minimum size requirement; for example, a course must have an enrollment of at least 75% of the room capacity into which it is scheduled. This avoids the problem associated with scheduling a course of section of 20 into a room seating 100 in a space-constrained environment[JJDV89].

The soft constraints are given less penalties compared to hard constraints. The penalty values we have taken on the basis of our consideration of constraints are given as under:

- If any instructor is assigned more than one lecture at any hour of any day, penalty is 500 per extra lecture. High penalty is assigned because it is difficult to get feasible timetable if this constraint is violated.
- If any lecture of any course is overlapping on a already filled timeslot, penalty is 200. Here also it is difficult to get feasible timetable if this

constraint is violated, but still one or two lectures may be rescheduled so penalty is less than the previous constraint.

- If any instructor is assigned lectures in consecutive hours on the same day, penalty is 200. Penalty of this constraint is decided as per the requirement of the university.

- If more than 6 lectures/week are assigned to any instructor, then penalty is 100 per additional lecture. This penalty is also to be decided by university requirements.

- If all the three lectures of any course are scheduled on Tuesday, Wednesday, Thursday i.e. not following the decided scheme 1 or scheme 2, penalty is 100. Penalty is less here as though it is violating the specified allocation schemes1 or scheme 2, but still it is better than not allocating the lecture at all.

To illustrate our solution methodology we have taken 3 test problems of different size shown in Table 2.1.

The above problems have been tackled using three different solution methods. We have developed a Heuristic Search method with random tie breaking and also did the mathematical formulation using standard linear integer programming method[Tri84]. Finally we developed the technique using GA and comparison has been done to make meaningful conclusions.

Table 2.1: Test Problems

Problem No.	Working Days/week	Rooms	Timeslots/day	Total Instructors
1	5	4	4	8
2	5	8	4	16
3	5	16	4	64

2.2 Literature Review

During last twenty years many contributions related to timetabling have appeared[Car86]. Recently, there has been a lot of attention paid to the problem of automating the construction of university course and examination timetables[Joh90]. One reason for this may be the huge variety of problems which are included in the field of timetabling; another reason is the fact that educational methods are changing, so the models have to be modified[AF89]. Finally computing facilities are widely available, so as a consequence the approach of timetabling has to take this phenomenon into account.

So far most successful computerized methods have been developed for special cases such as examination scheduling and timetabling. Although progress has been made, solving large instances is difficult. Consequently, course scheduling problms requiring multiple resourses, meeting time selection are usually solved by iterating between timetabling and single resourse assignment subproblems.

For example, Ferland and Roy[FR85] and Aubin and Ferland[AF89] descibe iterative room–time and class–time assignments systems. Carter[Car89] documents an iterative approach used at the University of Waterloo to develop course schedules.

A two-phased approach is proposed by Tripathy[Tri84]. He formulated the problem as a large integer linear programming probem in 0–1 variables. All the variables are described integer–valued. As the size of problem so formulated is very large, hence variables are redefined after grouping operation to reduce the size of the problem. In grouping operation, first students groups are formed with the condition that a group contains students enrolled in the same scheme. The formation of student groups forms the basis for formation of subject groups. He defines the subject group as group of subjects which are attended by the same set of student groups, and lastly the rooms are grouped on the basis of capacity of rooms. A room group is associated with each subject group as seating capacity required is known because of association of subject group with student groups.

He has used Lagrangian relaxation approach[Tri80], i.e. relaxing some of the constraints of the IP problem and solving the associated Lagrangian problem which is an IP problem and gives better bounds than other relaxations i.e. relaxing the integrality conditions itself. Further for solving the relaxed problem he used "Out of Kilter" method [BW68], which is better than solving by linear programming method in terms of the size of problem

data and computation time. However he fails to get complete feasibility, so Branch and Bound [LD60] has been incorporated into the solution process. The problems he faced with the proposed method are large storage requirements and long computation time.

Other course scheduling strategies have relied on interactive human aid. J. M. Mulvey had devised a Network based optimizing approach [Mul82] expanding the solution strategy of faculty/course scheduling problem presented by Dyer and Mulvey[DM76] to include the classroom/time assignment model. The general solution strategy proposed by him contains the following components:

1. Generate the data required for approximate model. This data includes faculty, course and time information, all modified to reflect administrative policies.

2. (Approximation) Generate the network model and solve.

3. (Evaluation) Determine whether the candidate schedule is a feasible alternative. If not, return to step 1.

4. (Evaluation) Determine whether the candidate schedule is acceptable. If not, go to step 6.

5. Print the schedule.

6. Decide whether to persist in attempting to improve the candidate schedule by hand. If not, go to step 3.

7. (Modification) Make manual changes in the candidate schedule. Go to step 3.

8. (Modification) Make changes in the network formulation. Go to step 3.

He tested the algorithm on small test problems and concluded that since the complete enumeration of the 0–1 integer problem was not accomplished, so could not find the quantitative measure of degree of optimality.

Dinkel et al.[JJDV89] report an application based on Mulvey's model involving 300 sections, 20 rooms and 16 timeslots. Solution was accomplished with commercial integer programming code. A similar approach has recently been described by White and Wong [WW88] that allows incremental interactive time and room assignment with heuristic solution methods. Acceptable scheduled were reported for department level use.

Werra D.[Wer85] had given a comprehensive review of existing methodologies with emphasis on graph theeoretical model. In his work, first he describes the basis class-teacher model without including all constraints which are usually present in real cases. He described C as set of classes and T as set of teachers, where a class was defined as a set of students who follow exactly the same program and defined X_{ijk} to be 1 if class c_i and teacher t_j

meet at period k and 0 otherwise. He further increased the size of the problem and also constraints. Further he discussed the graph colouring methods and mathematical programming approach also. He concluded that the field of timetabling was a nice topic for a researcher in OR for several reasons: There were many problems to be solved which include pure combinatorial mathematics, computer scienc(data basis, management, implementation of algorithms etc.) and OR (constructing a good timetable might be viewed a multicriterion decision problem or a group decision problem in a human context).

A method for examination scheduling using Genetic Algorithms has been proposed by Ayhan Ergul[Erg95]. Timetable is represented in the chromosome by allocating one gene to every exam, i.e. every gene holds the slot number that the exam has been scheduled to, using adaptive mutation operator. The fitness of individual has been calculated on the basis of predetermined penalty points. However he has not taken room consideration into account.

Alan M. Abraham had developed a simple Heuristic framework for course timetabling but solving only small sized problem effectively[BB78]. H. E. Mausser and others used Annealed Neural Network approach for the timetabling problem, but the selection of suitable values for penalty parameters was critical for obtaining good solution using Neural Network approach. Thus the selection of good parameter values remain an area for further

study[HEMM96].

In a review by Kathryn A. Dowsland[Kat96], the use of Genetic Algorithms for solution of OR problems has been compared with other Heuristic techniques, such as Simulated Annealing, Tabu Search etc. The auther concluded that working with GA might give good results compared to other heuristic techniques[MI94].

A growing number of researchers are now turning to Evolutionary Algorithms as a powerful method of solving difficult timetabling problems, as Evolutionary Algorithms appear efficient in the area of scheduling problems[SP91], particular in their ability to consider and optimize the wide variety of different constraints that may be found in universities.

2.3 Proposed Methodologies

Discrete optimization problems have traditionally been solved using problem specific heuristic and mathematical techniques such as integer programming. Recently several heuristics search methods have been introduced that derive from physical and natural sciences: Genetic Algorithms, Neural Networks etc., we have used a **GA Approach** and compared its performance with conventional heuristic approach and linear integer programming method. The identical test problems have been used for comparing all the three methods. The three different methods are:

1. Heuristic Assignment.

2. Linear Integer Programming model.

3. GA Approach.

2.3.1 Heuristic Assignment

In this **Heuristic Assignment** the courses are allotted to different time slots with "first come first serve" formula. Since we have already stated that for every course two options for room and two options for instructor are available, so here while scheduling the course, we choose one option of room out of $r1$ and $r2$ randomly and similarly the decision for instructor is also done out of the two options $i1$ and $i2$ randomly. After the random selection, if it found that the room or instructor selected is already scheduled at that time, so another option is taken for room or instructor as the case may be. After fixing the room and instructor, the selection of allocation schemes, out of scheme 1(Monday Wednesday Friday) and scheme 2(Tuesday Thursday Monday/Friday) is also done randomly for scheduling all the lectures of that particular course. The data of filled slots, rooms and assigned instructors has been maintained to avoid overlapping of courses at the same time and in same room, and consecutive course assignments to instructors as far as possible.

The algorithm for Heuristic Assignment is as follows:

STEP 1: Randomly decide the preference order of room and faculty for any course, from given options i.e. $r1\&r2$ and $i1\&i2$.

STEP 2: Assign course to instructor $i = i1$.

STEP 3: Randomly select the preference order for lectures allocation scheme 1(Monday Wednesday Friday) or 2(Tueesday Thursday Friday/Monday) and start checking the free timeslot first for r1 and then for r2 on both the schemes in order till get the free slot, put a flag and come out of search.

STEP 4: If not find a free slot in STEP 3, add penalty in fitness of the solution and put the error message, "Course Not Scheduled".

STEP 5: Start from the day found in STEP 3, check for free slot, if same instructor is having other lecture on the same day and slot then assign this course to other option of instructor, if other is also having lecture in same slot on same day then try the room other than selected in STEP 3 and find the slot for this course as in STEP 3.

STEP 6: Allocate the lectures of the course according to either of the allocation schemes which is selected in STEP 3, and put the flags after allocating and penalties if not allocated.

STEP 7: If the course allocated is the last one, exit, else repeat the process.

After allocating all the courses the fitness of the solution is calculated adding all the penalties for constraints violations.

2.3.2 Linear Integer Programming Model

A standard Linear Integer Programming formulation is possible for this problem of course timetabling. The following notatons are used to derive the LIP formulation:

$c = 1, \ldots, C$, where C is total no. of courses to be scheduled.

$l = 1, \ldots, L$ where L is total no. of lectures in a week of a course

$r = 1, \ldots, R$ where R is total no. of classrooms available.

$i = 1, \ldots, I$ where I is total no. of instructors.

$d = 1, \ldots, D$ where D is total no. of working days/week.

$h = 1, \ldots, L$ where D is total timeslots available/day.

Since we have considered 5 working days/week, the value of D is taken as 5. And we have considered the total number of lectures of any course per week is 3, L is taken as 3 everywhere. Total timeslots available per day are considered only 4, the value of H is 4 everywhere.

P_{clridh} is the cost of assignment, depending on penalties.

X_{clridh} is a 0–1 variable, which is 1 for correct assignent of the lecture of any course to room, slot, instructr and day, else 0.

Y_{ci} is a 0–1 variable, which is 1 for right assignment of all the lectures of any course to desired instructor, else 0.

All the variables are declared as 0–1 variables.

Both the variables can be expressed as:

$$X_{clridh} = \begin{cases} 1 & \text{for right assignment} \\ 0 & \text{otherwise} \end{cases}$$

$$Y_{ci} = \begin{cases} 1 & \text{for right assignment} \\ 0 & \text{otherwise} \end{cases}$$

The objective is to minimize the the cost of assigning all the lectures of all the courses, the cost for right assignment is given as 1 and cost for undesirable assignments are given as per penalties. The objective function may be formulated as:

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1}^{D}\sum_{h=1}^{H} P_{clridh}X_{clridh} \tag{2.1}$$

The first and foremost conditon is that all the lectures of all the courses must be scheduled only once in whole week i.e. there should be only one right assignment in the whole week. It is formulated as:

$$\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} = 1 \quad \forall c = 1,\ldots,C \quad \forall l = 1,\ldots,L \tag{2.2}$$

The other condition required for the feasible timetable is that on any day, any hour, every room should be scheduled for atmost one lecture of any course.

This is formulated as:

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{i=1}^{I} X_{clridh} \leq 1 \quad \forall r = 1,\ldots,R \quad \forall d = 1,\ldots,D \quad \forall h = 1,\ldots,H \quad (2.3)$$

Another important condition is that on any day, any hour, every instructor can be scheduled for atmost one lecture of any course. This is formulated as:

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R} X_{clridh} \leq 1 \quad \forall i = 1,\ldots,I \quad \forall d = 1,\ldots,D \quad \forall h = 1,\ldots,H \quad (2.4)$$

Another constraint we have considered here is the maximum number of assignments, an instructor may have in a week. We have kept this limit as 6. This is formulated as:

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} \leq 6 \quad \forall i = 1,\ldots,I \quad (2.5)$$

The constraint we have formulated here, puts the limit on the number of lectures of any course on any day. We have considered that maximum one lecture of any course could be scheduled on any day.

$$\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{h=1}^{H} X_{clridh} \leq 1 \quad \forall c = 1,\cdots,C \quad \forall d = 1,\cdots,D \quad (2.6)$$

The following constraints put the limit on course scheduling. We have followed the two scheduling schemes for scheduling all the lectures of any course. In an effort to follow that, this constraint puts the limit of maximum of two lectures of any course in three consecutive days, i.e. Monday Tuesday Wednesday, Tuesday Wednesday Thursday and Wednesday Thursday Friday.

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,2,3}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=2,3,4}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=3,4,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \end{array}\right\} \quad \forall c = 1, \ldots, C \qquad (2.7)$$

In the effort to schedule all the lectures of any course as per decided scheduling schemes, this constraint puts the limit of maximum one lecture of any course on Tuesday Wednesday and Wednesday Thursday.

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=2,3}\sum_{h=1}^{H} X_{clridh} \leq 1 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=3,4}\sum_{h=1}^{H} X_{clridh} \leq 1 \end{array}\right\} \quad \forall c = 1, \ldots, C \qquad (2.8)$$

The following constraint puts the limit of maximum two lectures of any course to be scheduled on Monday Tuesday Friday or Monday Thursday Friday, all in the effort to follow the desired scheduling schemes.

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,2,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,4,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \end{array}\right\} \quad \forall c = 1,\ldots,C \qquad (2.9)$$

Another important constraint is that any instructor should not be scheduled to take classes at consecutive hours of any day. The following constraint reflects this condition.

$$\left.\begin{array}{l} \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=1,2} X_{clridh} \leq 1 \\ \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=2,3} X_{clridh} \leq 1 \\ \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=3,4} X_{clridh} \leq 1 \end{array}\right\} \quad \forall i = 1,\ldots,I \quad \forall d = 1,\ldots,D \quad (2.10)$$

This is an important constraint which ensures that all the lectures of any course should be assigned to one instructor only. The following two equations reflect this condition.

$$\sum_{i=1}^{I} Y_{ci} = 1 \quad \forall c = 1,\ldots,C \qquad (2.11)$$

$$\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} - LY_{ci} = 0 \quad \forall c = 1,\ldots,C \quad \forall i = 1,\ldots,I \quad (2.12)$$

All X_{clridh} and Y_{ci} are 0–1 variables.

The complete LIP formulation is given as under:

Minimize

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1}^{D}\sum_{h=1}^{H} P_{clridh}X_{clridh} \tag{2.13}$$

Such that:

$$\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} = 1 \quad \forall c = 1,\ldots,C \quad \forall l = 1,\ldots,L \tag{2.14}$$

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{i=1}^{I} X_{clridh} \leq 1 \quad \forall r = 1,\ldots,R \quad \forall d = 1,\ldots,D \quad \forall h = 1,\ldots,H \tag{2.15}$$

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R} X_{clridh} \leq 1 \quad \forall i = 1,\ldots,I \quad \forall d = 1,\ldots,D \quad \forall h = 1,\ldots,H \tag{2.16}$$

$$\sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} \leq 6 \quad \forall i = 1,\ldots,I \tag{2.17}$$

$$\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{h=1}^{H} X_{clridh} \leq 1 \quad \forall c = 1,\cdots,C \quad \forall d = 1,\cdots,D \tag{2.18}$$

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,2,3}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=2,3,4}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=3,4,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \end{array}\right\} \quad \forall c = 1,\ldots,C \tag{2.19}$$

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=2,3}\sum_{h=1}^{H} X_{clridh} \leq 1 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=3,4}\sum_{h=1}^{H} X_{clridh} \leq 1 \end{array}\right\} \quad \forall c = 1,\ldots,C \qquad (2.20)$$

$$\left.\begin{array}{l} \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,2,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \\ \sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{i=1}^{I}\sum_{d=1,4,5}\sum_{h=1}^{H} X_{clridh} \leq 2 \end{array}\right\} \quad \forall c = 1,\ldots,C \qquad (2.21)$$

$$\left.\begin{array}{l} \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=1,2} X_{clridh} \leq 1 \\ \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=2,3} X_{clridh} \leq 1 \\ \sum_{c=1}^{C}\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{h=3,4} X_{clridh} \leq 1 \end{array}\right\} \quad \forall i = 1,\ldots,I \quad \forall d = 1,\ldots,D \qquad (2.22)$$

$$\sum_{i=1}^{I} Y_{ci} = 1 \quad \forall c = 1,\ldots,C \qquad (2.23)$$

$$\sum_{l=1}^{L}\sum_{r=1}^{R}\sum_{d=1}^{D}\sum_{h=1}^{H} X_{clridh} - LY_{ci} = 0 \quad \forall c = 1,\ldots,C \quad \forall i = 1,\ldots,I \qquad (2.24)$$

$$X_{clridh} = \begin{cases} 1 & \text{for right assignment} \\ 0 & \text{otherwise} \end{cases}$$

$$Y_{ci} = \begin{cases} 1 & \text{for right assignment} \\ 0 & \text{otherwise} \end{cases}$$

All X_{clridh} and Y_{ci} are 0–1 variables.

The LIP model gives rise to large number of variable and constraints[Tri84]. For the problem which we have modelled here, the total number of variables

nv are :

$$nv = CLRIDH + CI \tag{2.25}$$

And the total number of constraints nc are :

$$nc = LC + RDH + IDH + I + CD + 7C + 3ID + CI \tag{2.26}$$

So if we consider for assigning 10 courses($C = 10$) in the problem size 1($D = 5, R = 4, L = 3, H = 4, I = 8$) as discussed earlier in this chapter, then the number of variables are 19280 and number of constraints are 598 using above equations. And for assignment of 15 courses($C = 15$) in the same problem, the number of variables are 28920 and number of constraints are 713. Also, as the number of courses or the size of problem increase, the number of variables and constraints increases.

For solving the problem with above formulation, the LP software "CPLEX" has been used on HP9000 UNIX machine. Results are discussed in next chapter.

2.3.3 GA Approach

To meet the need of a flexible and efficient technique to solve the course timetabling problem, we have developed a technique using Genetic Algorithms.

GAs can find solution to linear and nonlinear problems by simultaneously exploring multiple regions of the state space and exploiting promising areas through mutation, crossover and selection operations[Deb93]. Genetic Algorithms have proved to be an effective and flexible optimization tool that can find optimal or near optimal solutions. Unlike many optimization techniques, GAs do not make strong assumptions about the forms of the objective function [Mic92].

Most optimization methods maintain a single solution and improve it untill an optimal solution is found. GAs differ in that they maintain and manipulate a family, or population of solutions in the search for an optimal solution[Dav91]. GAs mimic the evolutionary process by implementing a "survival of fittest" strategy. In general the fittest individual of any population tend to reproduce and pass their genes to the next generations, thus improving the successive generations. However some of the worst individuals do, by chance, survive and reproduce but this helps in a way to maintain diversity of the population. A more complete discussion of GAs, including related topics can be found in Goldberg's book on GAs[Gol89a].

Each solution or individual in the population is described by a vector of variables(chromosome representation). The first step in GA procedure is to initialize the population by generating solutions randomly. Once the initial population is generated, each individual is evaluated by using the objective function to determine its fitness or value. "Selection" schemes are used to reproduction based on fitness of individuals. Their are various selection schemes available, like probabilistic selection which maintains the possibility of selecting the best individual many times for the mating pool, contrary to tournament selection where the number of copies of the best individual are restricted by tournament size, i.e. if tournament size is two(i.e. two individuals are tournamented for reproduction), then the best individual gets exactly two copies after reproduction. Individuals are selected in pair, randomly, to be parent for two new children[GD91]. There are various crossover operators available for the problems of different domains. In this operation crossover points are selected with some probability and children are created by concatenating the segments from both the parents. Each offspring inherits some information from each parent. After crossing, "mutation" operator probabilistically mutates all the bits of chromosomes with usually very low probability. This is helpful to bring some information which might be missing in the whole population.

We have also tried this problem with the *Elitist Model* as well, but the results are no good if we compare with the simple GA, of course this finding

is based on limited experimentation. In the *Elitist Model* the best individual of any genaration replaces the worst individual of next generation.

2.3.3.1 GA Solution Technique

In building a Genetic Algorithms based technique, five fundamental issues that affect the performance of GA must be settled. These are: Chromosome representation, Selection strategy, Genetic operators, Termination criterion and Evaluation measures.

2.3.3.2 Chromosome Representation

For any GA, a chromosome representation is needed to describe each individual in the population of interest. The reprsentation scheme determines how the problem is structured in GA, as well as the genetic operators which can be used. Each individual or chromosome is a sequence of "genes", the carriers of an individual's characterstics.

In the problem we have constructed the chromosome for full timetable, giving information about all the courses to be scheduled. The chromosome representation of our formulation involving n courses is as:

$$\text{Individual} \rightarrow (\underbrace{011\ldots101}_{X_1}\underbrace{110\ldots100}_{X_2}\ldots\underbrace{010\ldots111}_{X_n})$$

Here each X represents the information about one course, like X_1 for

course 1. Further breakup of X_1 is as:

$$X_1 \rightarrow \underbrace{0\ldots1}_{A_1}\ \underbrace{0}_{A_2}\ \underbrace{0\ldots0}_{A_3}\underbrace{1\ldots0}_{A_4}$$

where

A_1 On decoding gives the room number, for scheduling all the lectures of a particular course, and size of A_1 increases as the number of rooms.

A_2 suggests the day, i.e. if it is 1, scheme 1 for allocation of lectures would be followed else scheme 2.

A_3 On decoding gives the timeslot for lectures of this particular course.

A_4 On decoding gives the instructor selected, for this particular course, and the size of A_4 increases as the number of faculty.

We can also represent the length of chromosome required in terms of number of courses. Let r be the total number of rooms available, s be the total number of timeslots/day and i be the total number of instructors available. And if $r1$, $s1$ and $i1$ be the number of bits required to represent rooms, slots and instructors in a chromosome respectively, then :

$$r = 2^{r1}$$

$$\Rightarrow r1 = \log_2 r$$

$s = 2^{s1}$

$\Rightarrow s1 = \log_2 s$

$i = 2^{i1}$

$\Rightarrow i1 = \log_2 i$

and one bit is required for day code in total number of bits for a course so total number of bits required for one course:

$\log_2 r + \log_2 s + \log_2 i + 1$

so if total number of courses to be scheduled are n, then length of chromosome required would be:

$n(\log_2 r + \log_2 s + \log_2 i + 1)$

2.3.3.3 Selection Strategy

The selection scheme for reproduction plays a very important role in GA. We have used Tournament Selection strategy for reproduction, which ensures the two copies of best individual of the population after reproduction. In this selection we randomly shuffle the whole population, then if the tournament size is 2, we take pairs of 2 individuals from the top and tournament them. The winner on the basis of fitness goes to mating pool. As tournament size is

2, we have to do this process two times. Since, tournamenting is done twice, best individual will send two of its copies in mating pool.

2.3.3.4 Genetic Operators

Crossover and mutation are two other basic genetic operators. Crossover opeator combines information from two parents such that the two children have a resemblence to each parent. We have used here single point crossover which is least destructive in all the crossover operators, a feature useful in our problem. Mutation operator used here is uniform mutation. It toggles any bit depending on mutation probability. We have taken a slightly high mutation probability which really helps in reaching optimal solution by mutating the bit carrying room option or the bit carrying faculty option for a particular course. The general allowable values of p_c lie between 0.6 to 0.9 and allowable values for p_m lie between) 0.001 to 0.01, of course for better convergence parametrization is to be done to find optimal values of these probabilities.

2.3.3.5 Termination Criterion

The GA moves from one generation to another generation until a termination criterion is met, the most frequently used stopping criterion is a specified maximum number of generations. We have also used the same criterion in our approach.

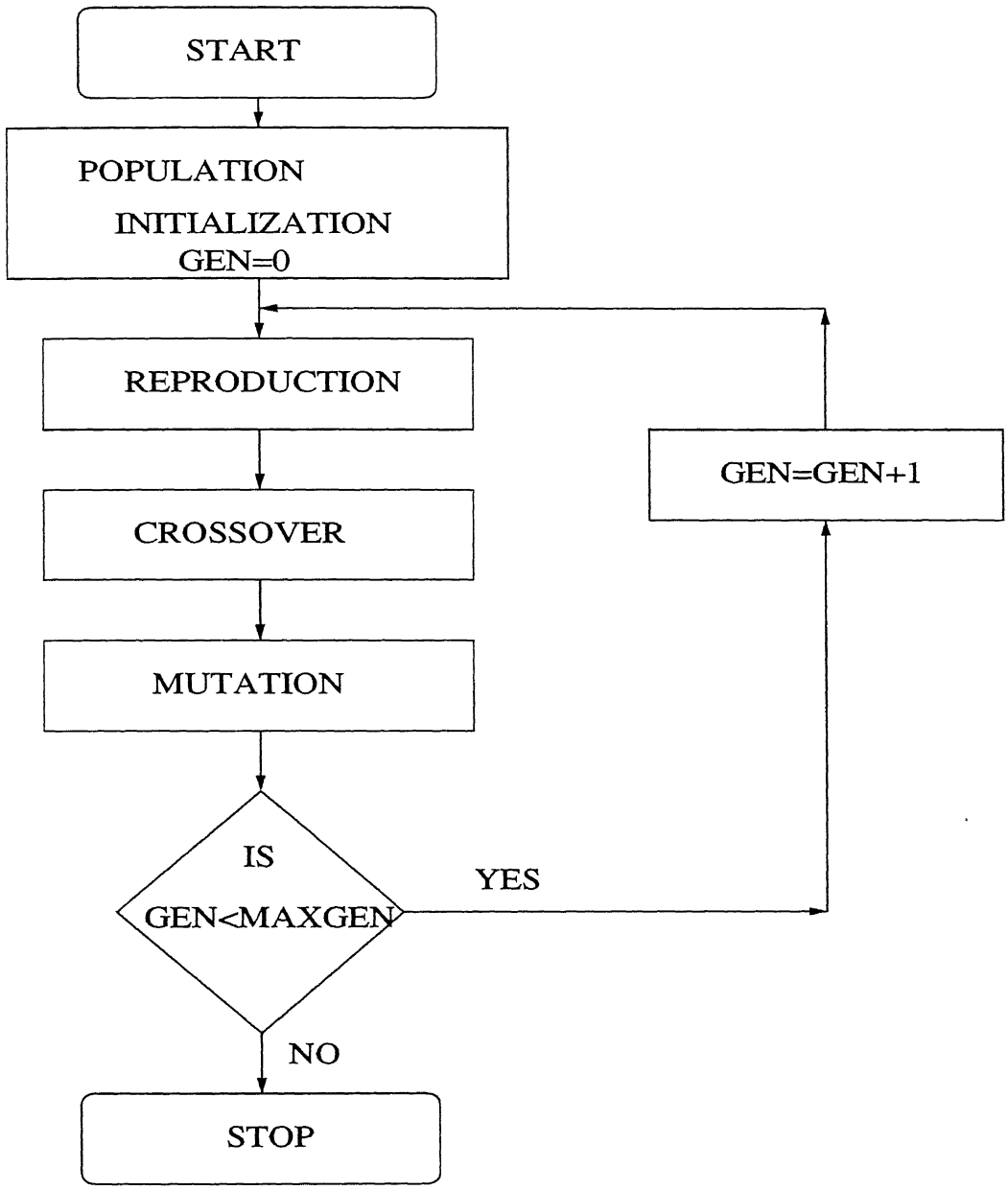
2.3.3.6 Evaluation Measure

Evaluation fuctions of many forms may be used in GA. We have calculated the fitness of an individual as a sum of predetermined penalties. In case of violation of constraints in course assignment scheme given by that individual, penalties are assigned for undesirable assignments of the courses. We have followed the same constraints and penalty structure for calculating fitness of an individual as we have done in other solution approaches(Heuristic and LIP) developed for making a good comparison on effectiveness of GA approach. In GA we have tried to minimize the fitness value over the generations. So the aim is to achieve the zero fitness. There may be many feasible schedules possibles but we are interested in finding only one feasible solution. For example, if we have two schedules with zero penalty, we'd prefer the one spreading the lectures all over the whole week evenly. But again this requirement may be achieved by penalising the schedules for that and hence pushing the GA to get the desired solution. Thus the evaluation measure is the only penalty term, we have taken.

The values of the penalties are also kept the same in all the models we have developed to make a realistic comparison of all the methods.

Results are discussed in the next chapter.

Figure 2.1: FLOW CHART FOR SIMPLE GENETIC ALGORITHM



Chapter 3

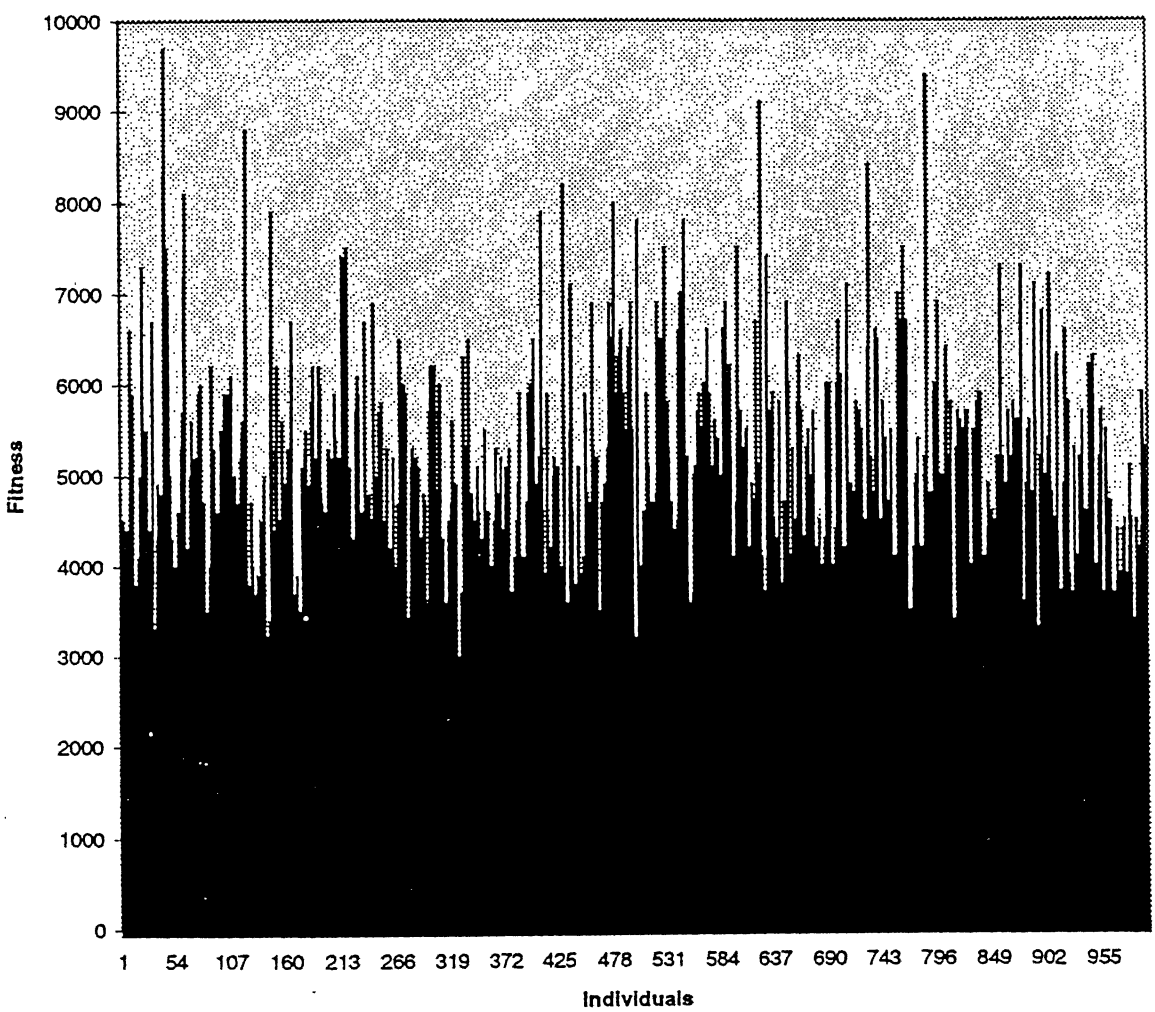
Results and Discussions

Evolutionary algorithms appear to show great promise for problems like course timetabling because of their ability to handle a variety of constraints which are normally found in such types of problems[SBK91]. By using an optimal combination of specially developed heuristic operators(crossover and mutation operators) the search can be guided towards the best regions of the solution space while using only a minimum amount of computation to reach optimal solution.

First we have plotted the fitness of the individuals randomly generated in the initial population of 1000. The Fig. 3.1 shows the minimum and maximum fitness of the individuals in initial random population. As, the fitness of the individuals in initial random population depends largly on the

random seed to generate the initial population, so we generated different populations with different random seed, but since the size of population we have taken is large enough so the average fitness of the individual in initial random population didn't vary much. As clear from the Fig. 3.1 that the fitness of good individuals initially generated in the population are of the order of 3000, and of the worst ones are of the order of even 9000.

Figure 3.1: Initial Random Generation

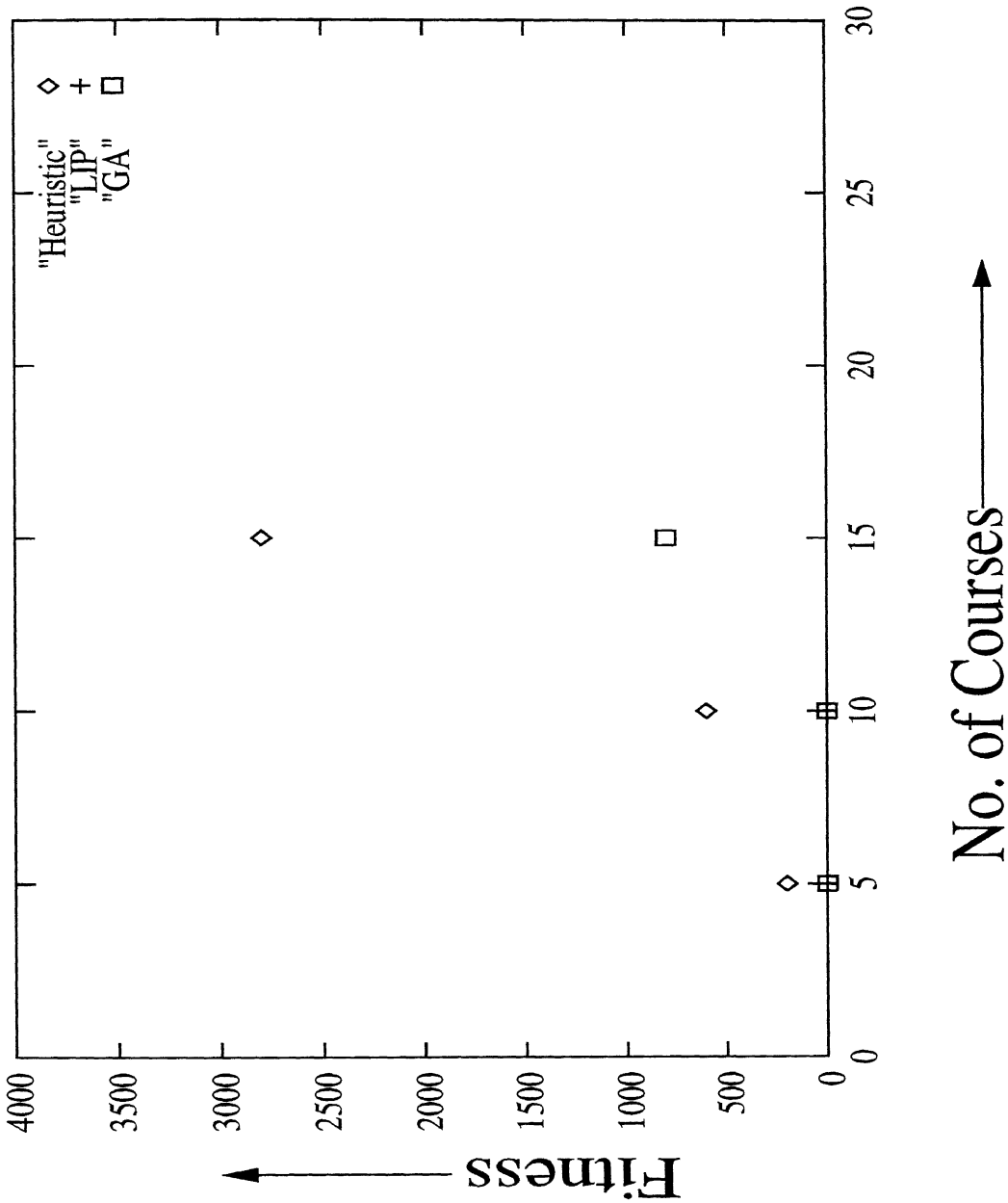


Results for the problem no.1 are shown in Fig. 3.2 . Fitness of the solutions obtained by Heuristic method, LIP method and GA approach, all are plotted in this figure. For the Heuristic Assignment and GA approach the best of the 10 run has been taken for each case of course assignments.

Fig.3.2 gives the results of the problem size 1(5 days, 4 rooms, 4 slots/day and available instructors 8). It is evident from the graph that Heuristic Assignment gives a mild penalty of 200 for the assignment of 5 courses and susequently gives a penalty of 600 for the assigning 10 courses and the penalty value finally reaches to a value of 2800 for the assignment 15 courses which is really a tight assignment within the framework of given constraints.

The mathematical model developed by linear integer programming technique generated the input file for the software "CPLEX" . The CPLEX has given the zero penalty solution for the assignment of 5 and 10 courses within the framework of similar constraints used for other methods but it could not solve the problem of assigning 15 courses under similar conditions at all, even after 436137 iterations. The problems of assignment of of 5 and 10 courses has taken 3.72, 1349.35 seconds and 89, 48072 iterations respectively to solve by this method. Also the another problem with this method is the size of the input data file, which has also been discussed by A. Tripathy[Tri84]. The increase in the size of the input data file is almost proportional to the increase in the number of courses. For example in our formulation with problem size 1,

Figure 3.2: Test Problem No. 1



the size of input data file for 5, 10 and 15 courses was of the order of 1.4Mb, 2.8 Mb and 4.4 Mb respectively. Since this modelling we did for the test problem with less number of constraints, so it appears that for real problem the size of input file for moderate number of courses will become cumbersome. At the end of this Chapter a comparison has been shown between the LIP method and GA approach.

We have also presented the results of GA approach in the same figure. The length of chromosome is $8c$, where c is the number of courses to be scheduled. The p_m and p_c values we have taken are 0.001 and 0.8 respectively. The GA approach is able to assign the courses 5 and 10 without any penalty, though with large population size and more number of generations i.e. with more computations but giving the assignment of 15 courses with penalty of 800(population size 200 and maximum no. of generations 200).

It has been found that the whole penalty to the solution was assignened because it was not able to find the right option for room and instructor for few courses. The probable reason for this may be Hamming Cliff[Gol89a], as the options for room or faculty for a particular course being maximally different binary sequences may be the cause this kind of problems. For example if we consider the options as 7 and 8 then with 4 bit coding, though 7 and 8 are very close in decoded space but their corresponding binary sequences being 0111 and 1000 are maximally different. These cases are difficult to search for binary coded GA. In such cases, optimal solution may be found with

high mutation probability. Also a large population size and large number of generations may give optimal results.

It is clear that in this approach, the number of bits representing one course increase with increase in the size of problem, or we can say because increase in the number of rooms and Instructors as we have already discussed in Chapter 2 in detail. So it is evident that length of chromosome would be increasing with the size of the problem. To overcome this problem, we have modified our GA approach. In this approach all the the parameters are same as in the previous GA approach, we discussed in Chapter 2, the only difference is in chromosome representation, so we now call this new approach as **Modified GA approach**. The chromosome representation is as:

$$\text{Individual} \rightarrow (\underbrace{01\ldots11}_{X_1}\underbrace{10\ldots00}_{X_2}\ldots\underbrace{00\ldots11}_{X_n})$$

Here each X represents the information about one course, like X_1 for course 1. The further breakup is as:

$$X_1 \rightarrow \underbrace{0}_{A_1}\ \underbrace{0}_{A_2}\ \underbrace{0\ldots0}_{A_3}\ \underbrace{1}_{A_4}$$

Where

A_1 This bit, being 0 or 1 gives the room number, for scheduling all the lectures of a particular course, out of the only two possible options for

a course given in input file.

A_2 suggests the day, i.e. if it is 1, scheme 1 for allocation of lectures would be followed else scheme scheme 2.

A_3 On decoding gives the timeslot for lectures of this particular course.

A_4 This bit, being 0 or 1 gives the faculty selected, for this particular course, out of the only two possible options for a course given in input file.

In this approach also each individual of the population of interest contains all the information about all the courses as in previous approach, the only difference is in the way information is stored. In this approach as the informations regarding room and instructor are stored in one bit each so the length of chromosome required is reduced considerably as compared to previous GA approach. So if s is total number of slots/day and $s1$ the number of bits required to represent these slots in total number of bits required for a course, then:

$$s = 2^{s1}$$

$$\Rightarrow s1 = \log_2 s$$

also, as one bit is required each for room, instructor and day in total number of bits of a course, total bits required for one course:

$\log_2 s + 1 + 1 + 1$

To schedule total n courses, the length of chromosome required is:

$n(\log_2 s + 3)$

The length of chromosome required with the previous GA approach discussed in last chapter is $n(\log_2 r + \log_2 s + \log_2 i + 1)$ where r and i are the total numbers of rooms and instructors available. It is found that length of chromosome in this modified approach is quite less if we compare with previous approach. The results for all the problems with this GA approach are presented in Fig. 3.3 to 3.5.

From the Fig. 3.3 it is clear that the problem of GA approach discussed earlier has been alleviated in this Modified GA approach, as it only takes the option of room and faculty by a single bit, being 0 or 1, thus paving way for smaller chromosome length and giving the optimal solution with smaller population size and in less number of generations, which we have also shown in factorial design. In the previous approach the rooms and instructors were selected out of the two options for the courses by decoding the corresponding bits in the chromosomes and it was found that major part of the penalty given to the solution was because of inability to find the correct options for rooms and instructors for few courses. It could be possible that by toggling few bits the desired solution could be found, but the chromosome length being very large, it may sometimes be difficult to get those desired mutations. The modified approach does not have this problem. Modified GA approach is able to give zero penalty solution for assignment of all 5, 10 and 15 courses. Here we have taken population size of 30 and p_c and p_m are taken 0.8 and 0.005 respectively. We have discussed in detail about the optimal parameters to be taken for best results. Hence, now on we compare the results of other methods with Modified GA approach only.

In the Fig. 3.4, now the competition is only between Heuristic Assignment and GA approach, as CPLEX could not solve the the problem size 1. Here again the Heuristic Assignment gives the penalties 1300, 2600, 4100 and 5600 for the assignment of 15, 20, 25 and 30 courses within the framework

Figure 3.3: Test Problem No. 1

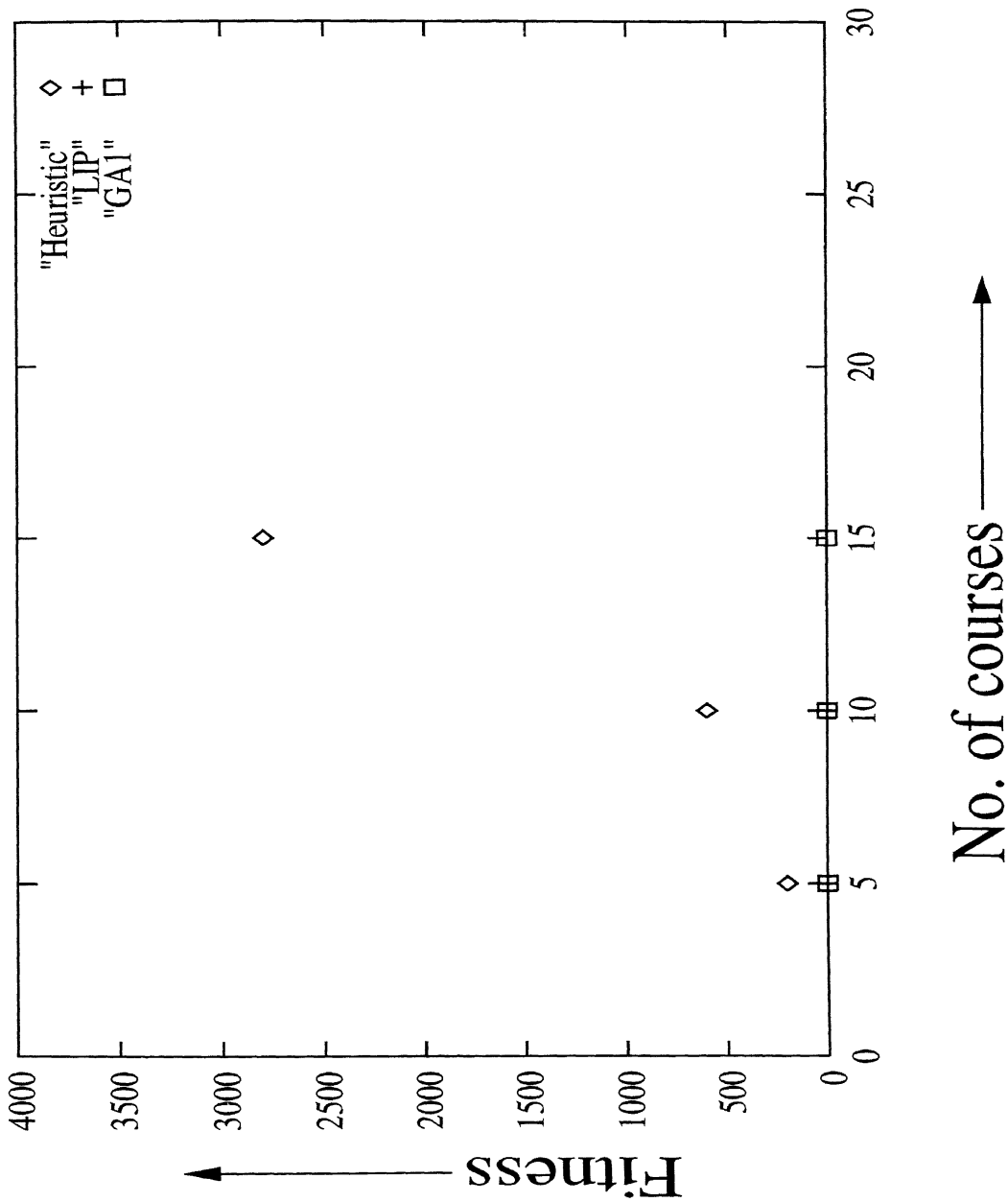


Figure 3.4: Test Problem No. 2

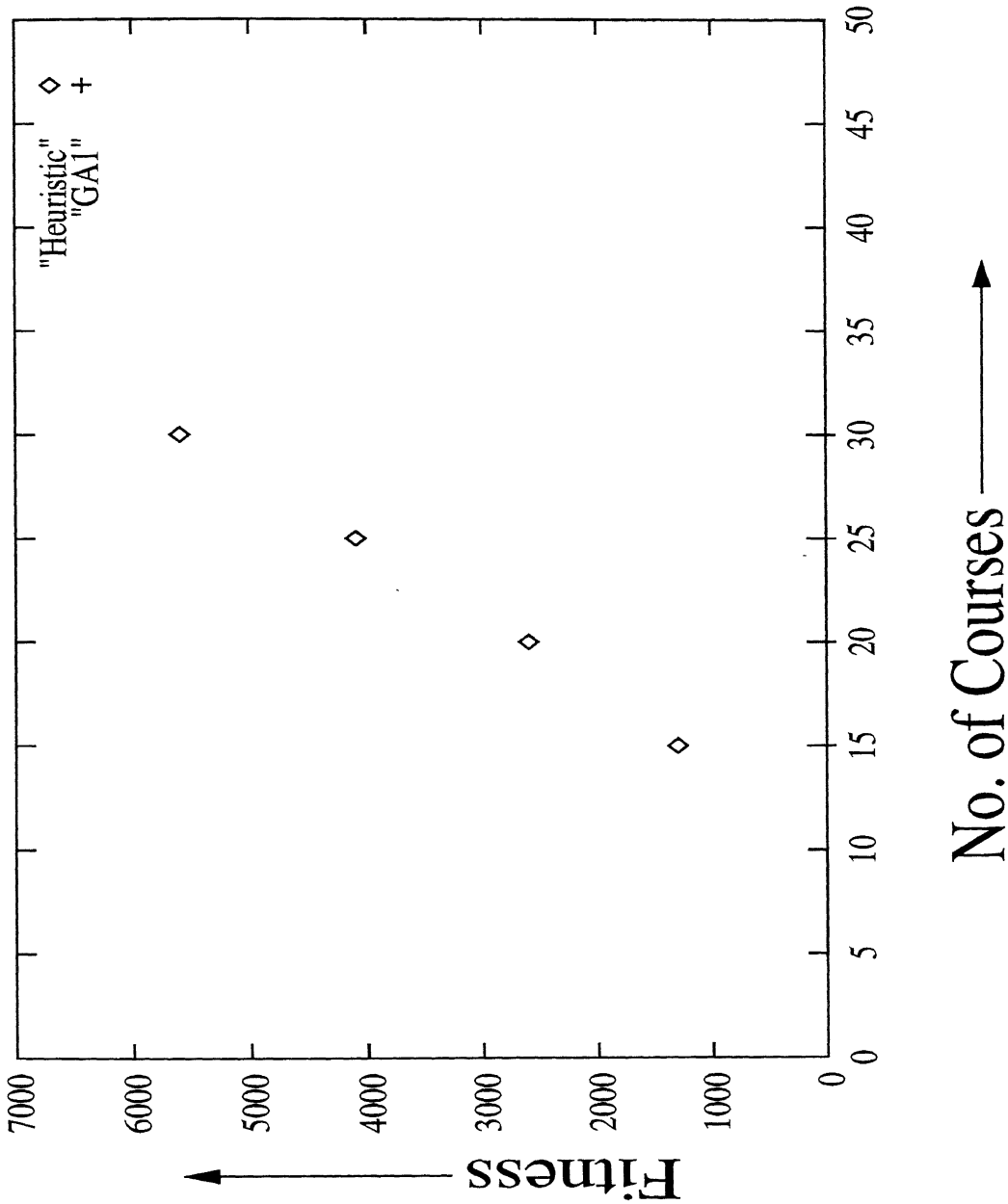
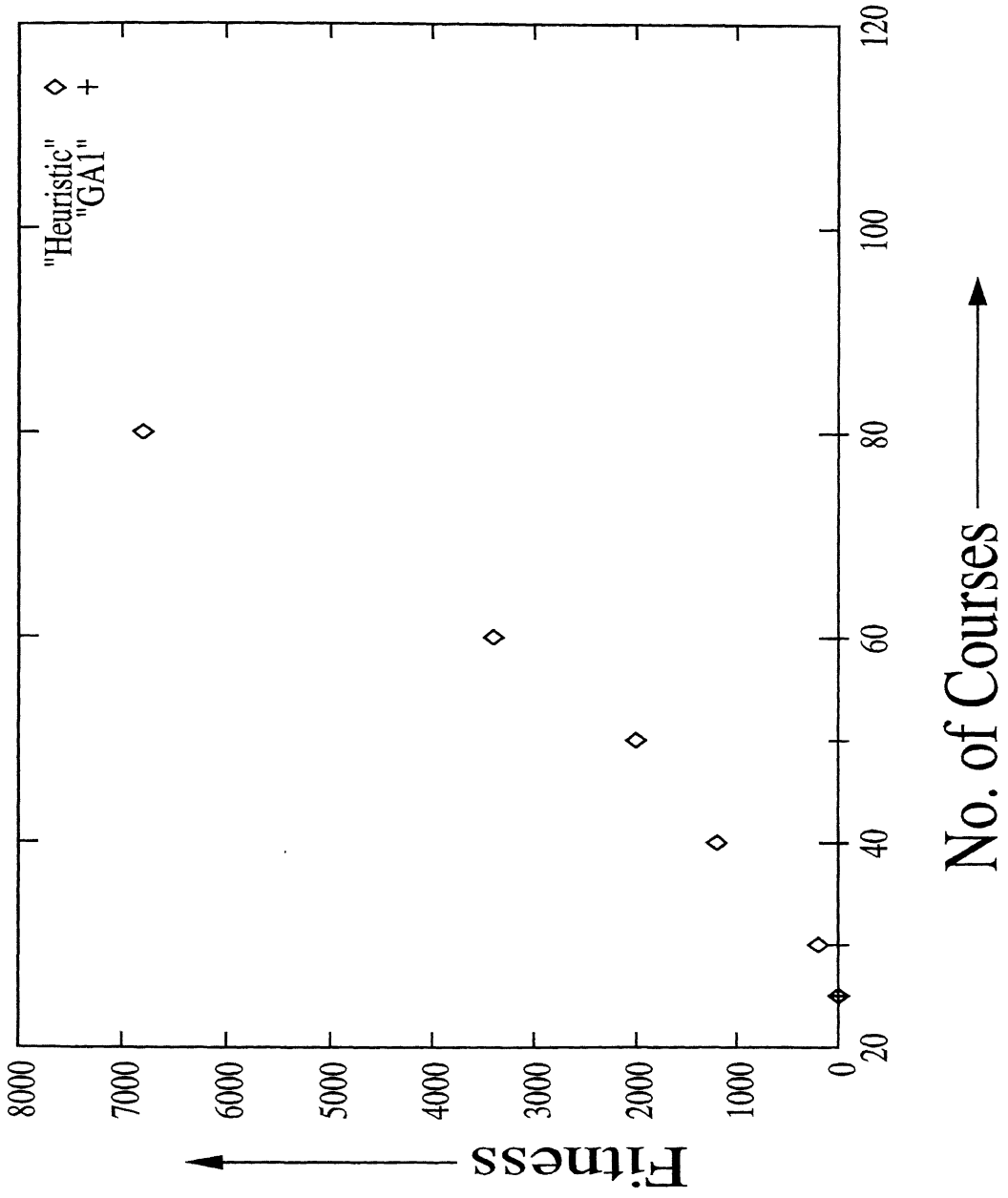


Figure 3.5: Test Problem No. 3



of problem size 2. And Here again the Modified GA approach has given zero penalty solution for the assignment of 15, 20, 25 and 30 courses under the conditions similar to other methods. Here the population size taken are 150, 150, 300 and 300 for 15, 20, 25 and 30 courses respectively. The p_c has been taken 0.8 in all cases and p_m are taken in the range of 0.005 to 0.01. As it is also found that optimal results are obtained with various different combinations of these GA parameters. We have discussed about this in the next section of this Chapter.

Finally the Fig.3.5, which shows the simulation results for the problem size 3. Here the Heuristic Assignment gives the zero penalty solution for the assignment of 25 courses as the slots available for the allotment of lectures are more due to problem size. But again as we move on for the assignment of more courses it again gives the penalties of 200, 1200, 2000, 3400 and 6800 for the assignment of 30, 40, 50, 60 and 80 courses. Here again the Modified GA approach maintains its performances, as it again gives zero penalty solution for the assignment of 25, 30, 40, 50, 60 and 80 courses. In this case also the assignment of 25, 30, 40, 50 and 60 were simple tasks for Modified GA approach. For example of 60 and 80 courses, we have taken the population size of 100 each with a low p_c and p_m of values 0.6 and 0.001 respectively. Again it is said that the optimal results are possible with many different combination of these GA parameters, which we have discussed in the next section of this Chapter.

3.1 Factorial Design

This is a big problem that what should be the population size, crossover probability and mutation probability to achieve the optimal solution with least computations[BDss]. So, for this, we have taken three different population size, crossover probabilities and mutation probabilities and we did experiments with 3^3 combinations of these parameters to show the dependence of obtained solution on GA paraneters. The results of experiments, we are presenting here and we have made recommendations on the basis of the results within the limit of our experiments.

These experiments were conducted on the problem size 1 discussed in Chapter 2 for 15 courses, with chromosome length of 75. The result of parametrization are presented in table 3.1.

The average fitness produced here is the average of 5 runs taken for each combination of population size, p_c and p_m taken here. On the basis of above results the average fitness for different p_c, p_m and population size are plotted in Fig.3.6 to Fig.3.8.

Table 3.1: Results of 3^3 combinations of population size, p_m and p_c

popsize	p_c	p_m	generations	Avg. Fitness
10	0.6	0.005	240	60
10	0.6	0.01	240	40
10	0.6	0.03	240	480
10	0.8	0.005	240	0
10	0.8	0.01	240	0
10	0.8	0.03	240	500
10	1.0	0.005	240	340
10	1.0	0.01	240	60
10	1.0	0.03	240	480
30	0.6	0.005	80	0
30	0.6	0.01	80	0
30	0.6	0.03	80	560
30	0.8	0.005	80	40
30	0.8	0.01	80	60
30	0.8	0.03	80	560
30	1.0	0.005	80	0
30	1.0	0.01	80	0
30	1.0	0.03	80	480
60	0.6	0.005	40	180
60	0.6	0.01	40	460
60	0.6	0.03	40	520
60	0.8	0.005	40	0
60	0.8	0.01	40	260
60	0.8	0.03	40	780
60	1.0	0.005	40	140
60	1.0	0.01	40	320
60	1.0	0.03	40	660

On the basis of the results of the factorial design experiments we can say that with increase in population size from 10 to 30, the solution improves. Also we observe that as population size is increased, solution does not improves because of limited number of generations so as we increase the number of generations, we get good results. It can be explained as, since we have supplied enough information to GA at chromosome level so the search for optimal solution may be very easy for Modified GA approach, hence it finds optimal solution in smaller population size. But on the basis of same number

Table 3.1: Results of 3^3 combinations of population size, p_m and p_c

popsize	p_c	p_m	generations	Avg. Fitness
10	0.6	0.005	240	60
10	0.6	0.01	240	40
10	0.6	0.03	240	480
10	0.8	0.005	240	0
10	0.8	0.01	240	0
10	0.8	0.03	240	500
10	1.0	0.005	240	340
10	1.0	0.01	240	60
10	1.0	0.03	240	480
30	0.6	0.005	80	0
30	0.6	0.01	80	0
30	0.6	0.03	80	560
30	0.8	0.005	80	40
30	0.8	0.01	80	60
30	0.8	0.03	80	560
30	1.0	0.005	80	0
30	1.0	0.01	80	0
30	1.0	0.03	80	480
60	0.6	0.005	40	180
60	0.6	0.01	40	460
60	0.6	0.03	40	520
60	0.8	0.005	40	0
60	0.8	0.01	40	260
60	0.8	0.03	40	780
60	1.0	0.005	40	140
60	1.0	0.01	40	320
60	1.0	0.03	40	660

On the basis of the results of the factorial design experiments we can say that with increase in population size from 10 to 30, the solution improves. Also we observe that as population size is increased, solution does not improves because of limited number of generations so as we increase the number of generations, we get good results. It can be explained as, since we have supplied enough information to GA at chromosome level so the search for optimal solution may be very easy for Modified GA approach, hence it finds optimal solution in smaller population size. But on the basis of same number

Figure 3.6: population size Vs. Avg. Fitness

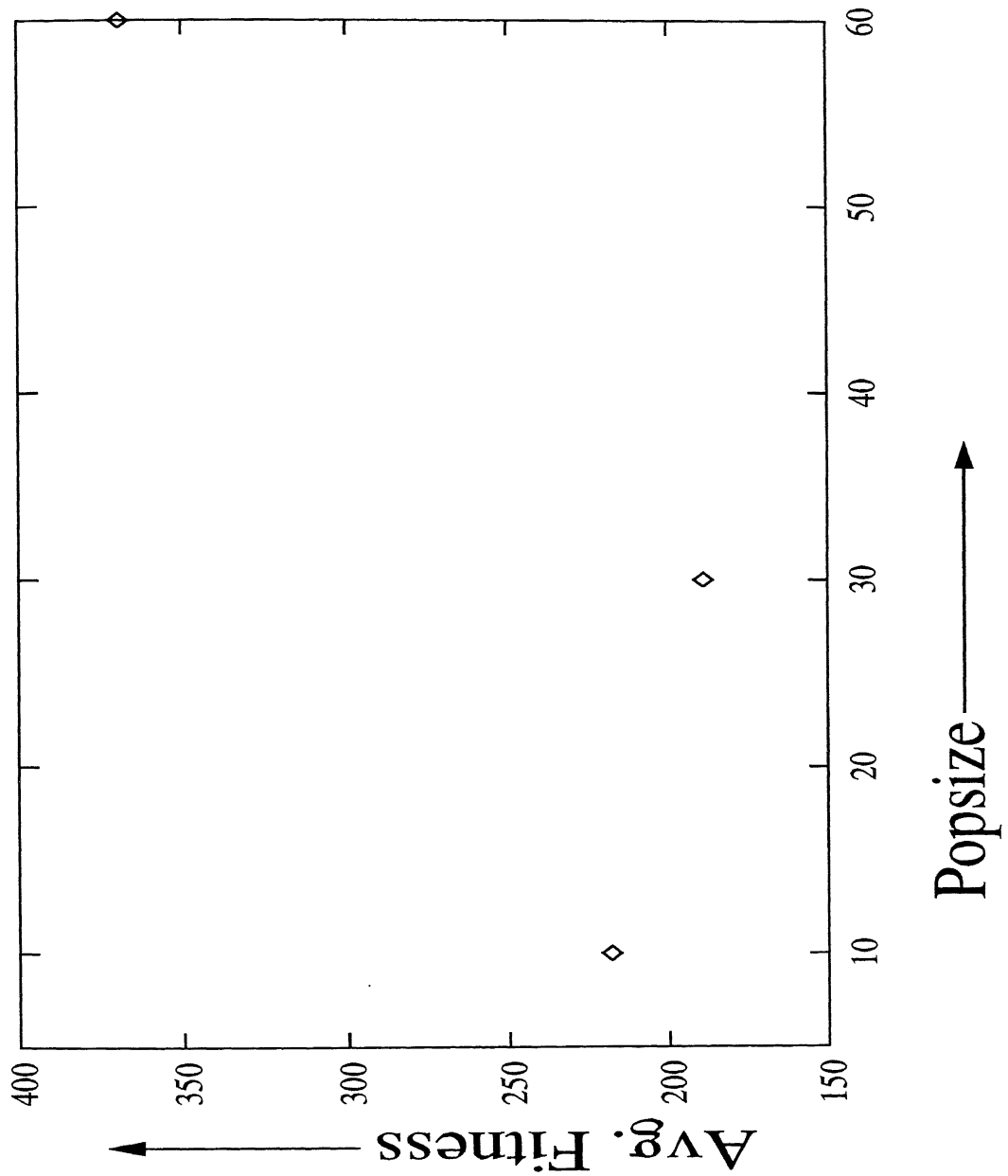


Figure 3.7: p_c Vs. Avg.Fitness

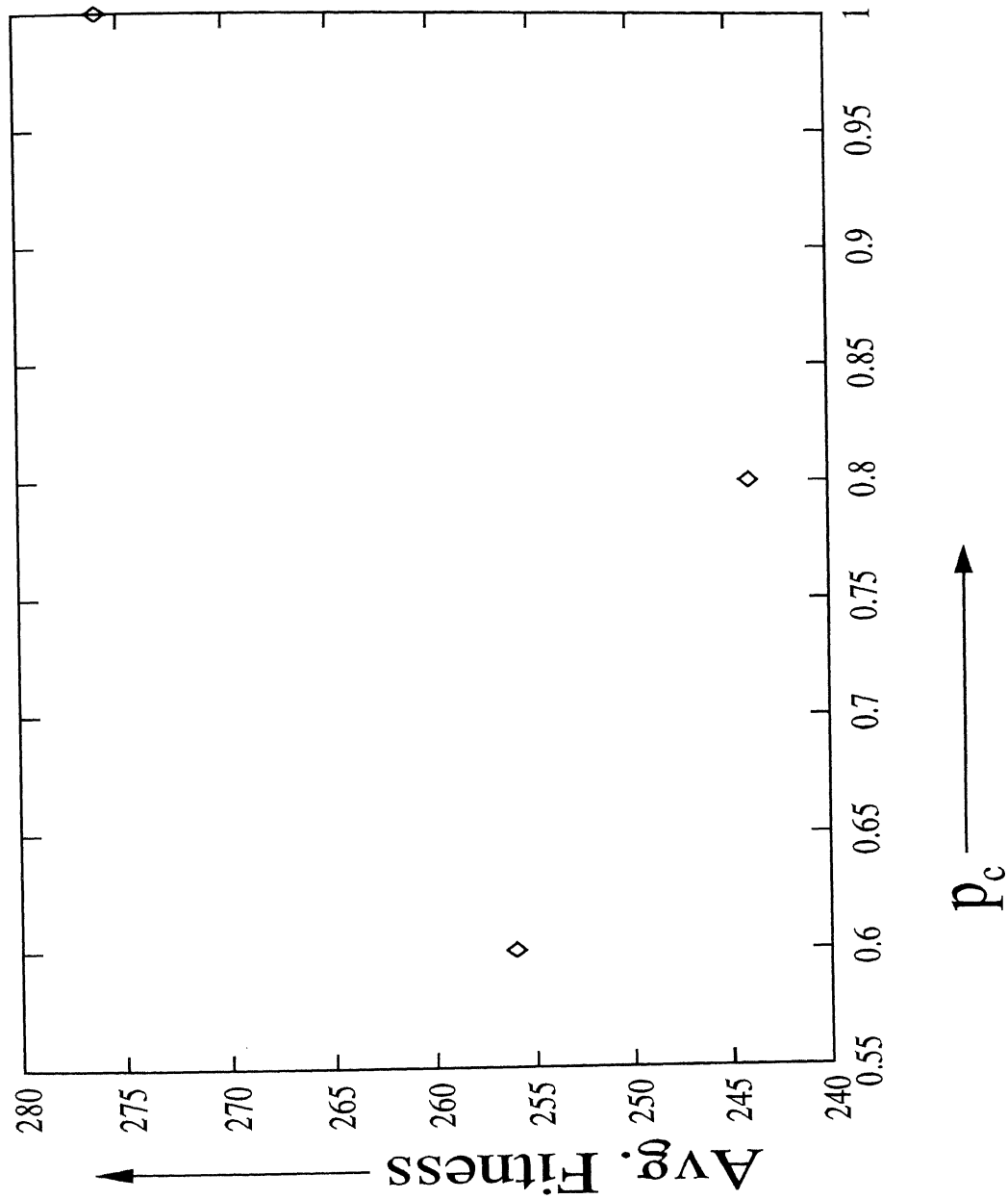
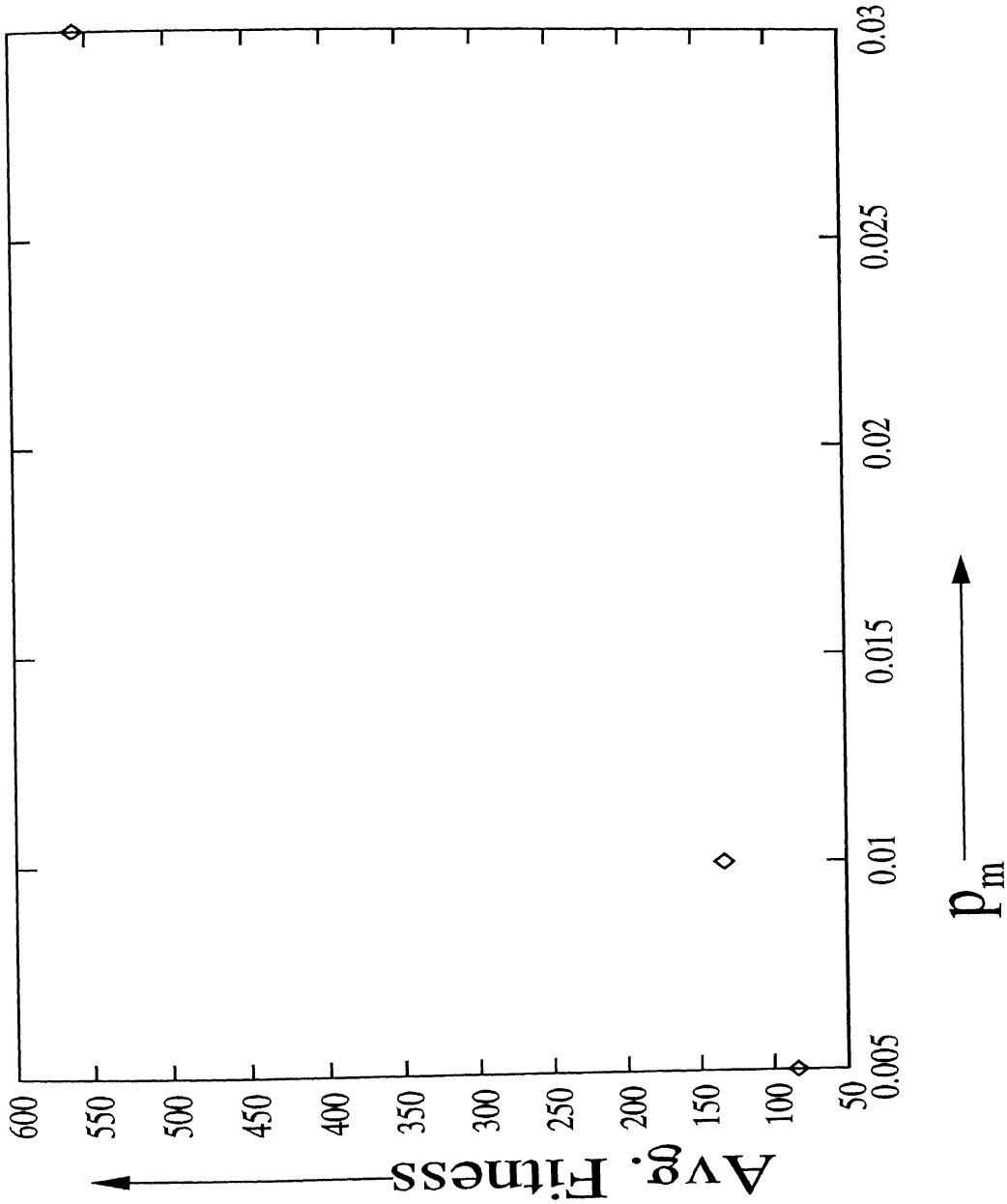


Figure 3.8: p_m Vs. Avg. Fitness



of function evaluations we do not get similar results with large population size because the number of generations we are running for large population size may be insufficient. For this, we tried the 60 population size with 80 generations(double of the previous run) and we got the results comparable with the ones with smaller population size and in few cases even better than that, few results are presented here.

In case of crossover probability, it was found that a moderate crossover probability(in this case 0.8) giving better results, and a low mutation probability was better option(in this case 0.005). Of course more experiments can be performed for optimal parametrization[BDss].

Here we have presented the comparison of LIP method and Modified GA approach on the basis of computation time taken to solve the same problems and also the number of iterations/generations required. Table 3.3 and 3.4 show the results. Further we have also shown the convergence of Modified GA approach irrespective of the initial random solution if using

Table 3.2: Results for population size 60, with large no. of gen.

popsize	p_c	p_m	generations	Best Fitness	Worst Fitness	Avg. Fitness
60	0.6	0.005	80	0	0	0
60	0.6	0.01	80	0	0	0
60	0.8	0.005	80	0	0	0
60	0.8	0.01	80	0	0	0
60	1.0	0.005	80	0	0	0
60	1.0	0.01	80	0	300	60

the optimal combination of GA parameters as discussed. In the Fig. 3.9 we have shown the convergence for the assignment of 15 courses in problem size 1(5days/week, 4rooms, 4slots/day and instructors available 8). We have solved the problem with 5 different initial random populations. The population size taken is 30 and maximum no. of generations are 60. p_m and p_c are taken 0.005 and 0.6 respectively.

Lastly we have presened the course assignments of dfferent problems we have discussed with all the methods discussed in Chapter 2.

Table 3.3: Computation Time for LIP method and Modified GA approach

courses	LIP Method time in sec.	Modified GA Approach time in sec.
5	3.72	0.02
10	1349.35	0.03
15	could not solve	0.05

Table 3.4: Iterations/Generations for LIP method and Modified GA approach

courses	LIP Method iterations	Modified GA Approach generations
5	89	in initial population
10	48072	2
15	could not solve	27

Figure 3.9: Convergence of GA using optimal parameters

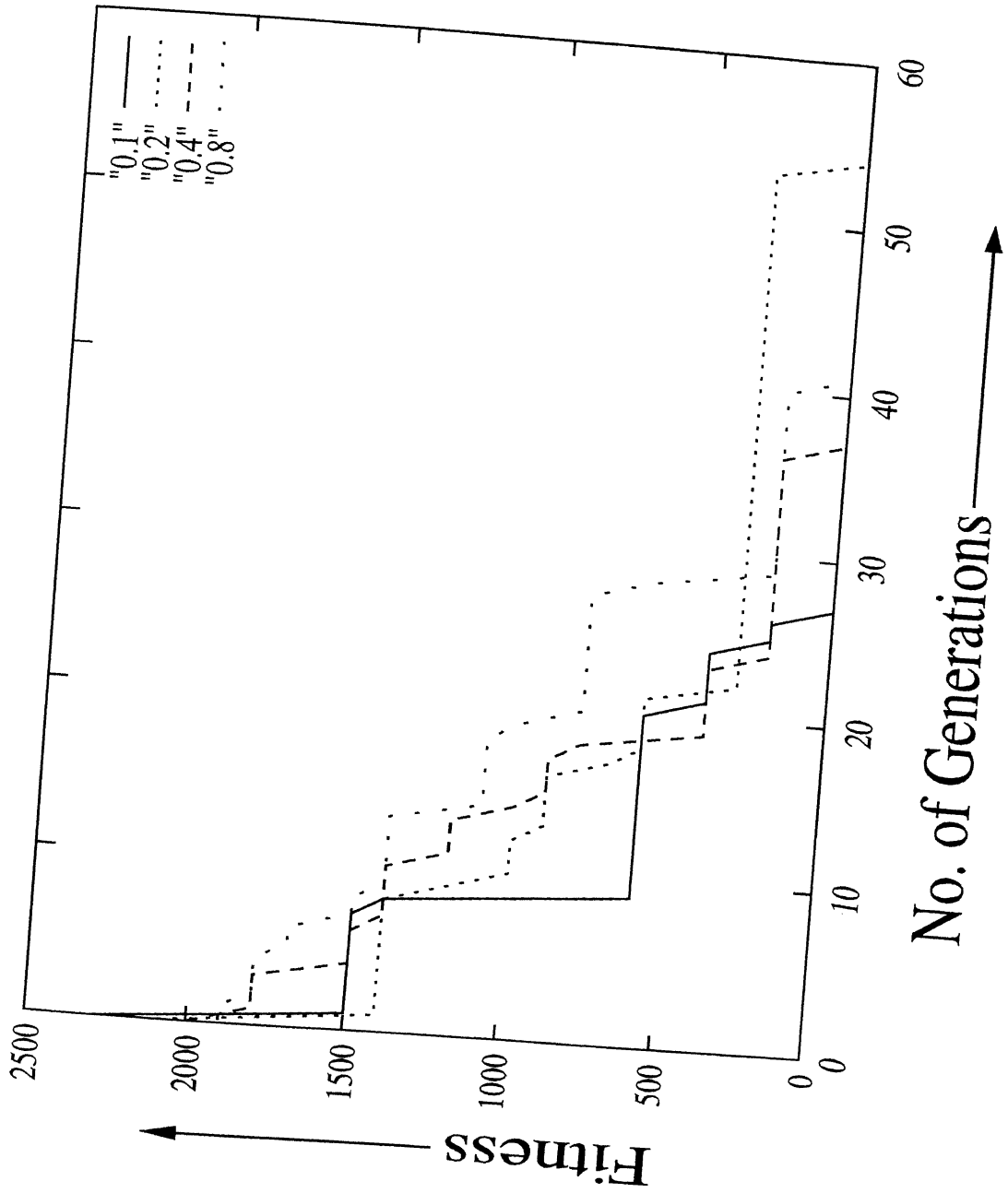


Table 3.5: Heuristic Approach for 5 courses(Problem No. 1) Fitness 200

	9–10	10–11	11–12	12–13
MON	C4 INS 7 L2 0 C2 INS 3 L3 0 C1 INS 1 L4 0	C5 INS 5 L3 0		
TUE	C3 INS 5 L1 0 C4 INS 7 L2 0			
WED	C2 INS 3 L3 0 C1 INS 1 L4 0	C5 INS 5 L3 0		
THUR	C3 INS 5 L1 0 C4 INS 7 L2 0			
FRI	C3 INS 5 L1 0 C2 INS 3 L3 0 C1 INS 1 L4 0	C5 INS5 L3 200		

Table 3.6: Modified GA Approach for 5 courses(Problem No. 1) Fitness 0

	9–10	10–11	11–12	12–13
MON	C4 INS 7 L2 0 C3 INS 6 L4 0			C2 INS 4 L3 0
TUE	C1 INS 2 L1 0 C4 INS 7 L2 0 C5 INS 5 L3 0 C3 INS 6 L4 0			
WED				C2 INS 4 L3 0
THUR	C1 INS 2 L1 0 C4 INS 7 L2 0 C5 INS 5 L3 0 C3 INS 6 L4 0			
FRI	C1 INS 2 L1 0 C5 INS 5 L3 0			C2 INS 4 L3 0

Table 3.7: GA Approach for 5 courses(Problem No. 1) Fitness 0

	9–10	10–11	11–12	12–13
MON	C5 INS 8 L4 0	C3 INS 5 L1 0		C2 INS 4 L4 0
TUE	C1 INS 1 L4 0		C4 INS 7 L3 0	
WED	C5 INS 8 L4 0	C3 INS 5 L1 0		C2 INS 4 L4 0
THUR	C1 INS 1 L4 0		C4 INS 7 L3 0	
FRI	C4 INS 7 L3 0 C5 INS 8 L4 0	C3 INS 5 L1 0 C1 INS 1 L4 0		C2 INS 4 L4 0

Table 3.8: LIP Method for 5 courses(Problem No. 1) Fitness 0

	9–10	10–11	11–12	12–13
MON	C4 INS 8 L2 0 C3 INS 5 L4 0	C2 INS 3 L2 0	C1 INS 1 L1 0	C5 INS 3 L3 0
TUE	C2 INS 3 L2 0 C3 INS 5 L4 0	C1 INS 1 L1 0 C4 INS 8 L2 0		C5 INS 3 L3 0
WED				
THUR	C3 INS 5 L1 0 C5 INS 3 L3 0	C4 INS 8 L2 0	C1 INS 1 L1 0 C2 INS 3 L2 0	
FRI				

Table 3.9: Heuristic Assignment for 10 courses(Problem No. 1) Fitness 600

	9–10	10–11	11–12	12–13
M O N	C10 INS 2 L1 0 C4 INS 1 L2 0 C2 INS 3 L3 0	C6 INS 6 L2 0 C5 INS 3 L3 200	C7 INS 8 L2 0 C8 INS 7 L3 0	C9 INS 4 L2 0
T U E	C3 INS 8 L1 0 C9 INS 4 L2 0 C5 INS 3 L3 0 C1 INS 6 L4 0	C10 INS 2 L1 0 C6 INS 6 L2 200 C8 INS 7 L3 0	C7 INS 8 L2 0	
W E D	C4 INS 1 L2 0 C2 INS 3 L3 0			
T H U R	C3 INS 8 L1 0 C9 INS 4 L2 0 C5 INS 3 L3 0 C1 INS 6 L4 0	C10 INS 2 L1 0 C6 INS 6 L2 200 C8 INS 7 L3 0	C7 INS 8 L2 0	
F R I	C3 INS 8 L1 0 C4 INS 1 L2 0 C2 INS 3 L3 0 C1 INS 6 L4 0			

Table 3.10: Modified GA Approach for 10 courses(Problem No.1) Fitness 0

	9–10	10–11	11–12	12–13
M O N	C8 INS 5 L1 0 C4 INS 6 L2 0 C2 INS 3 L3 0 C5 INS 4 L4 0	C1 INS 2 L1 0 C7 INS 1 L2 0	C3 INS 8 L1 0 C6 INS 6 L4 0	C10 INS 2 L1 0 C9 INS 1 L3 0
T U E	C4 INS 6 L2 0	C1 INS 2 L1 0	C7 INS 1 L2 0	C3 INS 8 L1 0 C5 INS 4 L4 0
W E D	C8 INS 5 L1 0 C2 INS 3 L3 0		C6 INS 6 L4 0	C10 INS 2 L1 0 C9 INS 1 L3 0
T H U R	C4 INS 6 L2 0	C1 INS 2 L1 0	C7 INS 1 L2 0	C3 INS 8 L1 0 C5 INS 4 L4 0
F R I	C8 INS 5 L1 0 C2 INS 3 L3 0		C6 INS 6 L4 0	C10 INS 2 L1 0 C9 INS 1 L3 0

Table 3.11: GA Approach for 10 courses(Problem No.1) Fitness 0

	9–10	10–11	11–12	12–13
MON		C6 INS 8 L3 0	C7 INS 6 L1 0	C4 INS 7 L2 0 C10 INS 3 L3 0 C5 INS 1 L4 0
TUE	C1 INS 1 L1 0 C3 INS 5 L2 0	C9 INS 2 L4 0	C2 INS 4 L2 0	C8 INS 7 L3 0
WED		C6 INS 8 L3 0	C7 INS 6 L1 0	C4 INS 7 L2 0 C10 INS 3 L3 0 C5 INS 1 L4 0
THUR	C1 INS 1 L1 0 C3 INS 5 L2 0	C9 INS 2 L4 0	C2 INS 4 L2 0	C8 INS 7 L3 0
FRI	C1 INS 1 L1 0 C2 INS 4 L2 0 C8 INS 7 L3 0 C9 INS 2 L4 0	C3 INS 5 L2 0 C6 INS 8 L3 0	C7 INS 6 L1 0	C4 INS 7 L2 0 C10 INS 3 L3 0 C5 INS 1 L4 0

Table 3.12: LIP Method for 10 courses(Problem No.1) Fitness 0

	9–10	10–11	11–12	12–13
MON	C1 INS 1 L1 0 C2 INS 4 L3 0	C4 INS 7 L2 0 C9 INS 5 L4 0	C10 INS 4 L2 0 C6 INS 3 L3 0	C8 INS 7 L1 0 C7 INS 1 L4 0
TUE	C5 INS 2 L1 0 C4 INS 7 L2 0	C3 INS 6 L3 0 C7 INS 1 L4 0	C6 INS 3 L3 0 C2 INS 4 L4 0	
WED	C8 INS 7 L1 0		C10 INS 4 L1 0	C1 INS 1 L1 0 C9 INS 5 L4 0
THUR	C4 INS 7 L2 0 C7 INS 1 L3 0	C3 INS 6 L1 0	C5 INS 2 L1 0 C6 INS 3 L2 0 C2 INS 4 L3 0	
FRI	C10 INS 4 L1 0 C8 INS 7 L2 0	C1 INS 1 L1 0 C5 INS 2 L4 0	C3 INS 6 L1 0 C9 INS 5 L3 0	

Table 3.15: Heuristic Approach for 30 courses(Problem No.2) Fitness 5600

	9–10	10–11	11–12	12–13
M	C16 INS 2 L1 0 C4 INS 1 L2 0	C18 INS 5 L1 0 C12 INS16 L2 **200**	C17 INS14 L2 0	C19 INS 1 L2 0
O	C2 INS 3 L3 0	C9 INS 4 L3 0	C13 INS13 L3 0	C24 INS11 L3 **200**
N	C1 INS16 L4 0	C10 INS 2 L4 **200**	C30 INS11 L4 **200**	
D	C15 INS15 L5 0	C5 INS 3 L5 **200**	C20 INS 3 L5 **200**	
A	C6 INS 6 L6 0	C14 INS11 L6 0	C28 INS15 L6 0	
Y	C7 INS 8 L7 0 C8 INS 7 L8 0	C21 INS 6 L7 **200**	C22 INS 8 L7 0	
T	C3 INS 8 L1 0 C4 INS 1 L2 0	C18 INS 5 L1 0 C26 INS12 L2 0	C25 INS15 L1 0	
U	C13 INS13 L3 0	C24 INS11 L3 **200**		
E	C10 INS 2 L4 0	C11 INS 9 L4 0		
S	C20 INS 3 L5 0			
D	C14 INS11 L6 0			
A	C21 INS 6 L7 0	C29 INS16 L7 0		
Y	C8 INS 7 L8 0	C23 INS 7 L8 **200**	C27 INS10 L8 0	
W	C16 INS 2 L1 0			
E		C12 INS16 L2 **200**	C17 INS14 L2 0	C19 INS 1 L2 0
D	C2 INS 3 L3 0	C9 INS 4 L3 0		
N	C1 INS16 L4 0		C30 INS11 L4 0	
E	C15 INS15 L5 0	C5 INS 3 L5 **200**		
S	C6 INS 6 L6 0		C28 INS15 L6 0	
D	C7 INS 8 L7 0		C22 INS 8 L7 0	
A				
T	C3 INS 8 L1 0	C18 INS 5 L1 0	C25 INS15 L1 0	
H	C4 INS 1 L2 0	C26 INS12 L2 0		
U	C13 INS13 L3 0	C24 INS11 L3 **200**		
R	C10 INS 2 L4 0	C11 INS 9 L4 0		
S	C20 INS 3 L5 0			
D	C14 INS11 L6 0			
A	C21 INS 6 L7 0	C29 INS16 L7 0		
Y	C8 INS 7 L8 0	C23 INS 7 L8 **200**	C27 INS10 L8 0	
F	C3 INS 2 L1 0 C16 L1 **200**	C25 INS15 L1 **200**		
R	C26 INS12 L2 0	C12 INS16 L2 **200**	C17 INS14 L2 0	C19 INS 1 L2 0
I	C2 INS 3 L3 0	C9 INS 2 L3 0		
D	C1 INS16 L4 0	C11 INS 9 L4 0	C30 INS11 L4 0	
A	C15 INS15 L5 0	C5 INS 3 L5 **200**		
Y	C6 INS 6 L6 0		C28 INS15 L6 **200**	
	C7 INS 8 L7 **500**		C22 INS 8 L7 0	
	C23 INS 7 L8 0			

Table 3.16: Modified GA Approach for 30 courses(Problem No.2) Fitness 0

	9–10	10–11	11–12	12–13
MONDAY	C10 INS10 L1 0 C17 INS14 L2 0 C13 INS15 L3 0 C18 INS 5 L4 0 C23 INS 7 L5 0 C22 INS 8 L6 0 C6 INS 6 L7 0 C27 INS12 L8 0	C1 INS 2 L1 0 C26 INS11 L2 0 C29 INS16 L4 0 C15 INS13 L5 0	C3 INS 8 L1 0 C19 INS 6 L2 0 C2 INS14 L3 0 C20 INS 3 L5 0 C21 INS 7 L7 0	C11 INS 9 L1 0 C24 INS11 L2 0 C4 INS 1 L3 0 C12 INS16 L5 0 C14 INS12 L6 0 C28 INS10 L7 0 C5 INS 4 L8 0
TUESDAY	C11 INS 9 L1 0 C16 INS 2 L4 0 C8 INS 5 L5 0	 C17 INS14 L2 0 C20 INS 3 L5 0 C7 INS 1 L6 0 C30 INS 4 L8 0	C1 INS 2 L1 0 C9 INS 9 L2 0	C3 INS 8 L1 0 C25 INS15 L4 0 C6 INS 6 L7 0 C27 INS12 L8 0
WEDNESDAY	C10 INS10 L1 0 C13 INS15 L3 0 C18 INS 5 L4 0 C23 INS 7 L5 0 C22 INS 8 L6 0	 C26 INS11 L2 0 C29 INS16 L4 0 C15 INS13 L5 0	C19 INS 6 L2 0 C2 INS14 L3 0 C21 INS 7 L7 0	C24 INS11 L2 0 C4 INS 1 L3 0 C12 INS16 L5 0 C14 INS12 L6 0 C28 INS10 L7 0 C5 INS 4 L8 0
THURSDAY	C11 INS 9 L1 0 C16 INS 2 L4 0 C8 INS 5 L5 0	 C17 INS14 L2 0 C20 INS 3 L5 0 C7 INS 1 L6 0 C30 INS 4 L8 0	C1 INS 2 L1 0 C9 INS 9 L2 0	C3 INS 8 L1 0 C25 INS15 L4 0 C6 INS 6 L7 0 C27 INS12 L8 0
FRIDAY	C10 INS10 L1 0 C9 INS 9 L2 0 C13 INS15 L3 0 C18 INS 5 L4 0 C23 INS 7 L5 0 C22 INS 8 L6 0 C30 INS 4 L8 0	C26 INS11 L2 0 C29 INS16 L4 0 C15 INS13 L5 0 C7 INS 1 L6 0	C19 INS 6 L2 0 C2 INS14 L3 0 C16 INS 2 L4 0 C8 INS 5 L5 0 C21 INS 7 L7 0	C24 INS11 L2 0 C4 INS 1 L3 0 C25 INS15 L4 0 C12 INS16 L5 0 C14 INS12 L6 0 C28 INS10 L7 0 C5 INS 4 L8 0

Chapter 4

Conclusions

On the basis of the results we have got on comparing the existing methodologies with our proposed Modified GA approach for the University Timetabling Problem, we made following conclusions.

1. Course Timetabling problem gives rise to a large search space as the number of constraints increase.

2. Genetic Algorithms (GAs) appear to be more effective than a Linear Integer Programming method and Heuristic Assignments, we have developed here for simple structured NP-Hard problems with large search space, because their population approach and use of only function values. Also time taken to solve such problems is also reasonable.

3. The additional advantage in using GA approach is because of their ability to handle a large variety of problem constraints without increasing complexity.

So, overall it can be concluded that the proposed Modified GA approach may be used to produce acceptable solutions. A more realistic model may be made and tested using the proposed approach. The success of GAs in this problem suggests their application in the domain of scheduling problems.

Bibliography

[AF89] J. Aubin and J. A. Ferland. A large scale timetabling problem. *Computers and Operations Research*, 16(1):67–77, 1989.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, New York, 1974.

[BB78] Alan A. Barham and John B.Westwood. A simple heuristic to facilitate course timetabling. *Journal of Operational Research Society*, 29:1055–1060, 1978.

[BDss] T. P. Bagchi and K. Deb. Calibration of ga parameters: The design of experiments approach. *Computer Science and Informatics Journal*, In Press.

[BW68] T. A. Bray and C. Witzgall. Algorithm 336 netflow(h). *Comm. ACM 11*, 9, 1968.

[Car86] M. W. Carter. A survey of practical applications of examinations timetabling algorithms. *Operations Research*, 34:193–202, 1986.

[Car89] M. W. Carter. A lagrangian relaxation approach to the classroom assignment problem. *Information Systems and Operational Research(INFOR)*, 27(2):230–246, 1989.

[Dav91] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York USA, 1991.

[DD96] Yip-Hoi Derek and Debasish Dutta. A genetic algorithm application for sequencing operation in process planning for parallel machining. *IIE Transactions*, 28:55–68, 1996.

[Deb93] K. Deb. *Genetic Algorithms for Engineering Design Optimization.* Proceedings of Advance Study Institute, IIT Madras, 1993.

[Deb95] K. Deb. *Optimization for Engineering Design: Algorithms and Examples.* Prentice Hall of India, 1995.

[DM76] J. Dyer and J. Mulvey. The implementation of an integrated optimization/information system for academic departmental planning. *Management Science*, 22:1332–1341, 1976.

[ELMP96] Ronald L. Rardlin Edward L. Mooney and W. J. Parmanter. Large scale classroom scheduling. *IIE Transactions*, 28:369–378, 1996.

[Erg95] A. Ergul. A genetic algorithm for university examination scheduling. *M.Sc. Thesis, Deptt. of Computer Engg., Middle East Technical University*, 1995.

[FR85] J. A. Ferland and S. Roy. Timetabling problem for university as assignment of activities to resourses. *Computers and Operations Research*, 12(2):207–218, 1985.

[GD91] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, pages 69–93, 1991.

[Gol89a] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesly, Reading Mass.,USA, 1989.

[Gol89b] D. E. Goldberg. Sizing population for serial and parallel genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79, 1989.

[HEMM96] Michall J. Magazine H. E. Mausser and John B. Moore. Application of an annealed neural network to a timetabling problem. *INFORMS Journal on Computing*, 8(2):103–117, 1996.

[JAJK96] C. Thomas Culberth Jefferey A. Joins and Russell E. King. Manufacturing cell design : An integer programming model employing genetic algorithms. *IIE Transactions*, 28:69–85, 1996.

[JJDV89] J. Mote J. J. Dinkel and M. A. Venkataramanan. An efficient decision support system for academic course scheduling. *Operations Research*, 37(6), 1989.

[Joh90] D. Johnson. Timetabling university examinations. *Journal of Operational Research Society*, 41:39–47, 1990.

[Kat96] Dowsland A. Kathryn. Genetic algorithms–a tool for or. *Journal of Operational Research Society*, 47:550–561, 1996.

[LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problem. *Econometrica*, 28:497–520, 1960.

[MI94] Tadahiko Murata and Hisao Ishibuchi. Performance evaluation of genetic algorithms for flow shop scheduling problems. *IEEE*, 4:812–817, 1994.

[Mic92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer – Verlag, New York, 1992.

[Mul82] J.M. Mulvey. A classroom time assignment model. *European Journal of Operations Research*, 9:64–70, 1982.

[SBK91] Yutaka Miyabe Sugata Bagchi, Serdar Uckun and Kazuhito Kawamura. Exploring problem specific recombination operators for job shop scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 10–17, 1991.

[SP91] Gilbert Syswerda and Jeff Palmucci. An application of genetic algorithms to resourse scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 502–508, 1991.

[ST93] A. E. Smith and D. M. Tate. Genetic optimization using penalty function. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 499–505, 1993.

[Tri80] A. Tripathy. A lagrangian relaxation approach to course timetabling. *Journal of Operational Research Society*, 31:599–603, 1980.

[Tri84] A. Tripathy. School timetabling – a case in large binary integer linear programming. *Management Science*, 30(12):1473–1489, 1984.

[Wer85] D. De Werra. An introduction to timetabling. *European Journal of Operations Research*, 19:151–162, 1985.

[WW88] G. M. White and K. S. Wong. Interactive timetabling in universities. *Computers in Education*, 12(4):521–529, 1988.

```
/*****************************************************************/
/* The GA approach developed for University Timetabling problem  */
/* has used the program "SGA.C" developed by D. E. Goldberg.     */
/* Only modified part of the program has been produced here, for */
/* remaining part readers may refer SGA.C available with         */
/* Dr. Kalyanmoy Deb, Deptt. of Mech. Engg., IIT Kanpur. His     */
/* email address is deb@iitk.ernet.in                            */
/*****************************************************************
#include<stdio.h>
#include "cox.h" /* File containing name of courses */
#include "tch.h" /* File containing name of teachers */
#include "Days.h"/* File containing name of working days */
#include "Room.h"/* File containing rooms available */
#define LINELENGTH 80     /*  width of printout */
#define BITS_PER_BYTE 8   /* number of bits per byte on this machine */
#define UINTSIZE (BITS_PER_BYTE*sizeof(unsigned))    /* # of
bits in unsigned */
#define  nc  15 /* No. of courses */
#define  nr  4 /*No. of classrooms available */
#define  nd   5 /* No. of working days per week */
#define  ns   4  /* No. of timeslots per day */
#define  ni  8 /* No. of instructors available */
#define  nbr  1 /*No. of bits for room in total bits of a course */
#define  nbi  1/*No.of bits for instructors in total bits of a course */
#define  nbs  2 /*No. of bits for timeslot in total bits of a course */

/* file pointers */
FILE *outfp, *infp;

/* Global structures and variables */
struct course
{
int cr;
int rm; /* room allocated for the course */
int sl; /* slot for the course */
int dy; /* day code, for the course */
int te; /* teacher allocated for the course */
};
struct course cor[nc];
struct individual
{

unsigned *chrom;/*chromosome string for the individual*/
```

```
struct course cor[nc];   /*  course informations */
double   fitness;      /* fitness of the individual */

int     AA[nd][nr][ns];  /*Info. about vacant slots */
int     BB[nd][nr][ns];  /*Info. for  additional course in  slots */
int     CC[nd][nr][ns];  /*Info. for instructors in different slots */
int     PP1[nd][nr][ns];  /*Penalty in all slots */

int     PP2[nd][nr][ns];  /*penalty in all the slots */
int     PP3[nd][nr][ns];  /*penalty in all the slots */
int     NN[nd][nr][ns];/*Info. for no.of extra courses in all slots */
int     pp2;          /* penalty */

int     F[ni];        /* Weekly assignment for instructor */
int       xsite;         /*  crossover site at mating */

int       parent[2];    /* who the parents of offspring were */

int       *utility;  /* utility field can be used as pointer to a */


/* dynamically allocated, application-specific data structure */
};
struct bestever
{
unsigned *chrom;  /* chromosome string for the best-ever individual */

struct course cor[nc];   /*  course informations */
double   fitness;    /* fitness of the best-ever individual */

int     AA[nd][nr][ns];  /*Info. about vacant slots */
int     BB[nd][nr][ns];  /*Info. for  additional course in  slots */
int     CC[nd][nr][ns];  /*Info. for instructors in different slots */
int     PP1[nd][nr][ns];  /*Penalty in all slots */

int     PP2[nd][nr][ns];  /*penalty in all the slots */
int     PP3[nd][nr][ns];  /*penalty in all the slots */
int     NN[nd][nr][ns];/*Info. for no.of extra courses in all slots */
int     pp2;          /*  penalty */

int    F[ni];      /* Weekly assignment for instructor */

int       node;        /* node from whence came best-ever */

int       generation; /* generation which produced it */
```

```
};

/* Application-dependent external declarations go after here...  */
double dshare,alpha;
int sampleshrsize, sampleshare;
/* sga.h - global declarations for main(), all variable
declared herein must  */
/*         also be defined as extern variables in external.h
!!!              */
/*-----------------------------------------------------------------
---------------*/

#define LINELENGTH 80                                        /*
width of printout */
#define BITS_PER_BYTE 8            /* number of bits per byte
on this machine */
#define UINTSIZE (BITS_PER_BYTE*sizeof(unsigned))    /* # of
bits in unsigned */
struct individual *oldpop;                  /* last generation
of individuals */

struct individual *newpop;                  /* next generation
of individuals */

struct bestever bestfit;                        /* fittest
individual so far */

double sumfitness;                     /* summed fitness for
entire population */

double max;                                  /* maximum
fitness of population */

double avg;                                  /* average
fitness of population */

double min;                                  /* minumum
fitness of population */

float  pcross;                                   /*
probability of crossover */

float  pmutation;                                 /*
probability of mutation */
```

```
int    numfiles;                                          /*
number of open files */

int    popsize;                                              /*
population size */

int    lchrom;                        /* length of the chromosome
per individual */

int    chromsize;              /* number of bytes needed to
store lchrom string */

unsigned    chromlen[nc];   /*chromosome length for each course */
int    gen;                                    /* current
generation number */

int    maxgen;                                 /* maximum
generation number */

int    run;                                       /*
current run number */

int    maxruns;                             /* maximum number
of runs to make */

int    printstrings = 1;     /* flag to print chromosome
strings (default on) */

int    nmutation;                             /* number of
mutations per node */

int    ncross;                              /* number of
crossovers per node */

int  i1, i,kk;
int crm[nc][2];                          /* penalty matrix for
course room matrix */
int cote[nc][2] ;                        /*penalty matrix for course
teacher matrix */
int check;                    /*variable for checking improvement in
fitness */

/*****************************************************************/
/** Here are included the functions which are modified as per    */
```

```
/*  the need of the problem.                                              */
/*                                                                        */
/**************************************************************************/



decodedvalue(chrom,value)
unsigned *chrom ;
int value[];
{
int k,j,stop,tp,bitpos,mask=1,position,bits_per_var,bits,nextcor;
int powa(), bitpow;

for(k=0;k<nc;k++) value[k]=0;
bits_per_var = chromlen[0];
bits = 0;
nextcor = 1;
position = 0;
for(k = 0; k < chromsize; k++)
{
if(k == (chromsize-1))
stop = lchrom-(k*UINTSIZE);
else
stop = UINTSIZE;
/* loop thru bits in current byte */
tp = chrom[k];
for(j = 0; j < stop; j++) {
if (bits >= bits_per_var) {
bits_per_var = chromlen[nextcor];
bits = 0;
position = nextcor;
nextcor++;
}
bits++;
if (bits_per_var > 0) {
/* test for current bit 0 or 1 */
if((tp&mask) == 1) {
bitpow = powa(2,(bits-1));
value[position] += bitpow;
}
tp = tp>>1;
}
else j--;
}
}
```

```
}

objfunc(critter)


struct individual *critter;
{
unsigned mask=1;    /* mask for current bit */

unsigned bitpos;    /* current bit position */

unsigned tp;
double fitn;
int powa(),bitpow,*temp_indv;
int r,t,j, k, stop,ll,j1;
int n = 2;
int f1=0;
int m1,m2,m3,k1,k2,k3,s,d,F[ni];
fitn = 0.0;
temp_indv=(int*)malloc(nc*sizeof(int));

decodedvalue(critter->chrom,temp_indv);

/*  initialize  the timetable matrix of nd days,and nr rooms
     and ns slots to 0*/
m1=m2=m3=0;
for (m1=0;m1<nd;m1++)
for(m2=0;m2<nr;m2++)
for(m3=0;m3<ns;m3++)
{
critter->AA[m1][m2][m3]=nc;
critter->BB[m1][m2][m3]=999;
critter->CC[m1][m2][m3]=ni;
critter->PP1[m1][m2][m3]=0;
critter->PP2[m1][m2][m3]=0;
critter->PP3[m1][m2][m3]=0;
critter->NN[m1][m2][m3]=0;
}


for(k=0;k<ni;k++)
critter->F[k]=0;

for (j=0;j< nc;j++)
critter->cor[j].cr=temp_indv[j];
```

```
for(i=0;i< nc;i++)
{
mask=1;
r=s=t=tp=d=0;
tp=critter->cor[i].cr;

/* Instructor Allocation */

if((tp&mask)==mask)
t=cote[i][0];
else
t=cote[i][1];

tp=tp>>1;


critter->cor[i].te=t;


/* Timeslot for the course */

for(j=0;j<nbs;j++)
{
if((tp&mask)==mask) s+=powa(2,j);
tp=tp>>1;
}

critter->cor[i].sl=s;

/* Day code to decide the allocation scheme 1 or scheme 2 */

if((tp&mask)==mask)
d=1;
else
d=0;

critter->cor[i].dy=d;
tp=tp>>1;


/* Room Allocation  */

if((tp&mask)==mask)
r=crm[i][0];
```

```
else
r=crm[i][1];

critter->cor[i].rm=r;

}
/* Course allocation starts here */

for (i=0;i<nc;i++)
{
int ll,m3,kk;
m3=ll=0;
d=critter->cor[i].dy;
r=critter->cor[i].rm;
s=critter->cor[i].sl;
t=critter->cor[i].te;

if(d==0)
{
for(j=0;j<nd;j=j+2)
{
ll=(j/2);
m3=2+ll;

if(critter->AA[j][r][s]==nc)
{
critter->AA[j][r][s]=i;
critter->CC[j][r][s]=t;
critter->F[t]+=1;
}
else
{
critter->NN[j][r][s]+=1;
critter->BB[j][r][s]=i;
f1+=200;
critter->PP1[j][r][s]+=200;
}
}
}
else
continue;
}

for(i=0;i<nc;i++)
{
```

```
int m3,ll;
m1=m2=j=ll=m3=0;
d=critter->cor[i].dy;
r=critter->cor[i].rm;
s=critter->cor[i].sl;
t=critter->cor[i].te;
if(d==1)
{
for(j=1;j<(nd-1);j=j+2)
{
ll=(j/2);
m3=2+ll;
if (critter->AA[j][r][s]==nc)
{
critter->AA[j][r][s]=i;
critter->CC[j][r][s]=t;
critter->F[t]+=1;
}
else
{
critter->NN[j][r][s]+=1;
critter->BB[j][r][s]=i;
f1+=200;
critter->PP1[j][r][s]+=200;
}
}

if (flip(0.5))
{
k1=0;
k2=4;
k3=2;
}
else
{
k1=4;
k2=0;
k3=2;
}

switch(0)
{
case 0:
for(j=0;j<ns;j++)
{
```

```
if(critter->AA[k1][r][j]==nc)
{
critter->AA[k1][r][j]=i;
critter->CC[k1][r][j]=t;
critter->F[t]+=1;
m1=1;
}
if(m1==1)break;
}
if(m1==1)break;

case 1:
for(j=0;j<ns;j++)
{
if(critter->AA[k2][r][j]==nc)
{
critter->AA[k2][r][j]=i;
critter->CC[k2][r][j]=t;
critter->F[t]+=1;
m1=1;
}
if(m1==1)break;
}
if(m1==1)break;

case 2:
for(j=0;j<ns;j++)
{
if(critter->AA[k3][r][j]==nc)
{
critter->AA[k3][r][j]=i;
critter->CC[k3][r][j]=t;
critter->F[t]+=1;
m2=1;
critter->PP1[k3][r][j]+=100;
f1+=100;
}
if(m2==1)break;
}
if((m1==0)&&(m2==0))
{
f1+=200;
critter->PP1[k3][r][j]+=200;
}
}
```

```
}
else
continue;
}


for(j=0;j<ni;j++)
if(critter->F[j]>6)
{
f1+=((critter->F[j])-6)*100;
critter->pp2=((critter->F[j])-6)*100;
}

/* Penalty for more than one lecture at the same time and day */

for(k=0;k<nd;k++) /*k represents no.of days in a week */
{
for(j=0;j<ns;j++) /*j is no. of time slots per day */
{
for(m1=0;m1<(nr-1);m1++)
{
for(m2=m1+1;m2<nr;m2++)
{
if((critter->CC[k][m1][j]!=ni)&&(critter->CC[k][m2][j]!=ni))
{
if((critter->CC[k][m1][j])==(critter->CC[k][m2][j]))
{
/* code folded from here */
f1+=500;
critter->PP2[k][m2][j]+=500;
/* unfolding */
}
}

}
}
}
}

/*Penalty if one teacher taking consecutive classes on the same day*/

for(k=0;k<nd;k++) /*k represents no.of days in a week */
{
for(j=0;j<nr;j++) /*j is no. of rooms available */
{
```

```
for(m1=0;m1<(ns-1);m1++)
{
for(j1=0;j1<nr;j1++)
{
for(m2=m1+1;m2<m1+2;m2++)
{
if((critter->CC[k][j][m1]!=ni)&&(critter->CC[k][j1][m2]!=ni))
{
/* code folded from here */
if((critter->CC[k][j][m1])==(critter->CC[k][j1][m2]))
{
f1+=200;
critter->PP3[k][j1][m2]+=200;
}
/* unfolding */
}
}
}
}
}
}
/* At this point, fitness = x */

/* fitn=(double)(f1);*/
fitn=1.0*f1;
critter->fitness=fitn;



}

/*************************************************************/
/*                                                           */
/*  Application Routine                                      */
/*************************************************************/

int powa(a,b)
int a,b;
{
int tempo=1;
int j;
if(b==0)
return(1);
else
{
for(j=1;j<=b;j++)
```

```
tempo=tempo*a;
return(tempo);
}
}
```

```
/*********************************************************/
/*      Heuristic Assignment                             */
/*********************************************************/

#include<stdio.h>
#include "cox.h" /* Name of courses */
#include "tch.h" /* Name of instructors */
#include "Days.h" /Working Days */
#include "Room.h" /* Rooms available */
#define  nc 5 /* No. of courses */
#define  nd 5 /* No. of working days per week */
#define  nr 4 /* No. of rooms available*/
#define  ns 4 /*No. of time slots per day*/
#define  ni 8  /* No. of instructors  available */

/* file pointers */
FILE *outfp, *infp;

/*-------------------------------------------------------------
---------------*/

int    numfiles;                                    /*
number of open files */

int  i1, i,kk,m1,m2,m3,f1,k,F[ni];
int crm[nc][2];                /*room options for courses */

int cote[nc][2] ;              /*instructor option  for courses */
int AA[nd][nr][ns] ;           /*Info. about all the  slot allocation */
int BB[nd][nr][ns] ;           /*Info. for extra courses */
int CC[nd][nr][ns] ;           /*Info. about all the instructors  */
int NN[nd][nr][ns] ;           /*Info. for no. of extra courses */
int PP[nd][nr][ns] ;           /*Penalty */



main(argc,argv)
int argc;
char *argv[];
{
FILE   *fopen();
/* determine input and output from program args */
numfiles = argc - 1;
switch(numfiles)
```

```
{
case 0:
infp = stdin;
outfp=stdout;
break;
case 1:
if((infp = fopen(argv[1],"r")) == NULL)
{
fprintf(stderr,"Input file %s not found\n",argv[1]);
exit(-1);
}
outfp = stdout;
break;
case 2:
if((infp = fopen(argv[1],"r")) == NULL)
{
fprintf(stderr,"Cannot open input file %s\n",argv[1]);
exit(-1);
}
if((outfp = fopen(argv[2],"w")) == NULL)
{
fprintf(stderr,"Cannot open output file %s\n",argv[2]);
exit(-1);
}
break;
default:
fprintf(stderr,"Usage is: sga [input file] [output file]\n");
exit(-1);
}


randomize();

app_data();

for(m1=0;m1<nd;m1++)
for(m2=0;m2<nr;m2++)
for(m3=0;m3<ns;m3++)
{
AA[m1][m2][m3]=nc;
PP[m1][m2][m3]=0;
NN[m1][m2][m3]=0;
BB[m1][m2][m3]=999;
CC[m1][m2][m3]=999;
}
```

```
/*F[t] indicates the lectures instructor t is taking in a week */
for(m1=0;m1<ni;m1++)
F[m1]=0;

for(i=0;i<nc;i++)
{
allocation();
}

/*Penalty for more than one course at a time for any instructor */

for(m1=0;m1<nd;m1++)
for(m2=0;m2<ns;m2++)
for(m3=0;m3<(nr-1);m3++)
for(i=m3+1;i<nr;i++)
{
if((CC[m1][m3][m2]!=999)&&(CC[m1][i][m2]!=999))
{
if(CC[m1][m3][m2]==CC[m1][i][m2])
{
f1+=500;
PP[m1][i][m2]+=500;
}
}
}

/*Penalty if one instructor taking consecutive classes on the same day*/

for(k=0;k<nd;k++) /*k represents no.of days in a week */
{
for(i=0;i<nr;i++) /*i is no. of rooms available */
{
for(m1=0;m1<(ns-1);m1++)
{
for(m3=0;m3<nr;m3++)
{
for(m2=m1+1;m2<m1+2;m2++)
{
if((CC[k][m3][m2]!=999)&&(CC[k][i][m1]!=999))
{
if(CC[k][i][m1]==CC[k][m3][m2])
{
f1+=200;
PP[k][m3][m2]+=200;
```

```
}
}
}
}
}
}
}

/* penalty for more than 6 lectures to any instructor in a week */

for(m1=0;m1<ni;m1++)
{
if(F[m1]>6)
f1+=(F[m1]-6)*100;
}

/*  for(m1=0;m1<nd;m1++)
   for(m2=0;m2<nr;m2++)
    for(m3=0;m3<ns;m3++)
      if(NN[m1][m2][m3]>0) f1+=300;*/

report();


}


/*-------------------------------------------------------------
--------------*/
/* app.c - application dependent routines, change these for
different problem */
/*-------------------------------------------------------------
--------------*/
app_data()
{
for(kk=0;kk<nc;kk++)
{
if(numfiles==0)
fprintf(outfp,"\nEnter crm elements for course %d",kk);
fscanf(infp,"\n%d %d",&crm[kk][0],&crm[kk][1]);
}
for(kk=0;kk<nc;kk++)
{
if(numfiles==0)
fprintf(outfp,"\nEnter cote elements for course %d",kk);
```

```
fscanf(infp,"\n%d %d",&cote[kk][0],&cote[kk][1]);
}
}

allocation()
{
int r1,r2,l,l1,l2,j,m1,m2,t,r,k1,k2,k3,jj,ll,t1,t2;
r1=r2=l=l1=l2=j=m1=m2=t=r=k1=k2=k3=jj=ll=t1=t2=0;

/* Selection of room and instructor for the course */

if(flip(0.5))
{
r1=crm[i][0];
r2=crm[i][1];
t1=cote[i][0];
t2=cote[i][1];
}
else
{
r1=crm[i][1];
r2=crm[i][0];
t1=cote[i][1];
t2=cote[i][0];
}

t=t1;

/* Search for timeslot  */

if(flip(0.5))
{
for(l1=0;l1<ns;l1++)
{
l=l1;
if(AA[0][r1][l1]==nc)
{
m1=1;
r=r1;
}
if(m1==1)break;
if(AA[0][r2][l1]==nc)
{
m1=1;
r=r2;
```

```
}
if(m1==1)break;
}
if(m1!=1)
{
for(l1=0;l<ns;l1++)
{
l=l1;
if(AA[1][r1][l1]==nc)
{
m2=1;
r=r1;
}
if(m2==1)break;
if(AA[1][r2][l1]==nc)
{
m2=1;
r=r2;
}
if(m2==1)break;
}
}
}


else
{
for(l1=0;l<ns;l1++)
{
l=l1;
if(AA[1][r1][l1]==nc)
{
m2=1;
r=r1;
}
if(m2==1)break;
if(AA[1][r2][l1]==nc)
{
m2=1;
r=r2;
}
if(m2==1)break;
}
if(m2!=1)
{
```

```
for(l1=0;l1<ns;l1++)
{
l=l1;
if(AA[0][r1][l1]==nc)
{
m1=1;
r=r1;
}

if(m1==1)break;
if(AA[0][r2][l1]==nc)
{
m1=1;
r=r2;
}

if(m1==1)break;
}
}
}

if((m1!=1)&&(m2!=1))

{
f1+=600;
fprintf(outfp,"\n course %d could not be allocated",i);
}


if(m1==1)

{
for(l1=0;l1<nr;l1++)
{
if(CC[0][l1][l]==t1) t=t2;
if(CC[0][l1][l]==t2)
{
for(j=0;j<ns;j++)
{

if((AA[0][r1][j]==nc)&&(j!=l))
{
l=j;
t=t2;
r=r1;
```

```
l1=1;
}
if(l1==1)break;

if((AA[0][r2][j]==nc)&&(j!=l))

{
l=j;
t=t2;
r=r2;
l2=1;
}
if(l2==1)break;
}
}
if(l1==1)break;
if(l2==1)break;
}

for(j=0;j<nd;j=j+2)
{
if(AA[j][r][l]==nc)
{
AA[j][r][l]=i;
PP[j][r][l]+=0;
CC[j][r][l]=t;
F[t]+=1;
}
else
{
BB[j][r][l]=i;
NN[j][r][l]+=1;
PP[j][r][l]+=200;
f1+=200;
}
}
}
if(m2==1)
{
for(ll=0;ll<nr;ll++)
{
if(CC[1][ll][l]==t1) t=t2;
if(CC[1][ll][l]==t2)
{
for(j=0;j<ns;j++)
```

```
{
if((AA[1][r1][j]==nc)&&(j!=l))
{
l=j;
t=t2;
r=r1;
l1=1;
}
if(l1==1)break;
if((AA[1][r2][j]==nc)&&(j!=1))
{
l=j;
t=t2;
r=r2;
l2=1;
}
if(l2==1)break;
}
}
if(l1==1)break;
if(l2==1)break;
}
for(j=1;j<(nd-1);j=j+2)
{
if(AA[j][r][l]==nc)
{
AA[j][r][l]=i;
PP[j][r][l]+=0;
CC[j][r][l]=t;
F[t]+=1;
}
else
{
BB[j][r][l]=i;
NN[j][r][l]+=1;
PP[j][r][l]+=200;
f1+=200;
}
}
if(flip(0.5))
{
k1=0;
k2=4;
k3=2;
}
```

```
else
{
k1=4;
k2=0;
k3=2;
}

switch(0)
{
case 0:
for(l1=0;l1<ns;l1++)
{
if(AA[k1][r][l1]==nc)
{
AA[k1][r][l1]=i;
CC[k1][r][l1]=t;
F[t]+=1;
jj=1;
}
if(jj==1)break;
}
if(jj==1)break;

case 1:
for(l1=0;l1<ns;l1++)
{
if(AA[k2][r][l1]==nc)
{
AA[k2][r][l1]=i;
CC[k2][r][l1]=t;
F[t]+=1;
jj=1;

}
if(jj==1)break;
}
if(jj==1)break;

case 2:
for(l1=0;l1<ns;l1++)
{
if(AA[k3][r][l1]==nc)
{
AA[k3][r][l1]=i;
CC[k3][r][l1]=t;
```

```
F[t]+=1;
jj=1;
PP[k3][r][l1]=100;
f1+=100;
}
if(jj==1)break;
}
}
if(jj!=1) f1+=200;
}
}
```

```
/*-------------------------------------------------------------
--------------*/
/* random.c - contains random number generator and related
utilities,          */
/* including advance_random, warmup_random, random, randomize,
flip, and rnd  */
/*-------------------------------------------------------------
--------------*/

#include <math.h>

/* variables are declared static so that they cannot conflict
with names of    */
/* other global variables in other files.  See K&R, p 80, for
scope of static */
static double oldrand[55];                    /* Array of 55
random numbers */

static int jrand;                                  /*
current random number */

static double rndx2;                          /* used with random
normal deviate */

static int rndcalcflag;                       /* used with random
normal deviate */


advance_random()
/* Create next batch of 55 random numbers */
{
```

```
int j1;
double new_random;

for(j1 = 0; j1 < 24; j1++)
{
new_random = oldrand[j1] - oldrand[j1+31];
if(new_random < 0.0) new_random = new_random + 1.0;
oldrand[j1] = new_random;
}
for(j1 = 24; j1 < 55; j1++)
{
new_random = oldrand [j1] - oldrand [j1-24];
if(new_random < 0.0) new_random = new_random + 1.0;
oldrand[j1] = new_random;
}
}


int flip(prob)
/* Flip a biased coin - true if heads */
float prob;
{
float randomperc();

if(randomperc() <= prob)
return(1);
else
return(0);
}


initrandomnormaldeviate()
/* initialization routine for randomnormaldeviate */
{
rndcalcflag = 1;
}


double noise(mu ,sigma)
/* normal noise with specified mean & std dev: mu & sigma */
double mu, sigma;
{
double randomnormaldeviate();

return((randomnormaldeviate()*sigma) + mu);
```

```
}


randomize()
/* Get seed number for random and start it up */
{
float randomseed;
int j1;

for(j1=0; j1<=54; j1++) oldrand[j1] = 0.0;
jrand=0;

if(numfiles == 0)
{
do
{
fprintf(outfp,"\n Enter random number seed, 0.0 to 1.0 -> ");
fscanf(infp,"%f", &randomseed);
} while((randomseed < 0.0) || (randomseed > 1.0));
}
else
{
fscanf(infp,"%f", &randomseed);
}


warmup_random(randomseed);
}


double randomnormaldeviate()
/* random normal deviate after ACM algorithm 267 / Box-Muller Method */
{
double sqrt(), log(), sin(), cos();
float randomperc();
double t, rndx1;

if(rndcalcflag)
{
rndx1 = sqrt(- 2.0*log((double) randomperc()));
t = 6.2831853072 * (double) randomperc();
rndx2 = sin(t);
rndcalcflag = 0;
return(rndx1 * cos(t));
}
```

```
else
{
rndcalcflag = 1;
return(rndx2);
}
}


float randomperc()
/* Fetch a single random number between 0.0 and 1.0 - Subtractive Method
/* See Knuth, D. (1969), v. 2 for details */
/* name changed from random() to avoid library conflicts on some machines
{
jrand++;
if(jrand >= 55)
{
jrand = 1;
advance_random();
}
return((float) oldrand[jrand]);
}


int rnd(low, high)
/* Pick a random integer between low and high */
int low,high;
{
int i;
float randomperc();

if(low >= high)
i = low;
else
{
i = (randomperc() * (high - low + 1)) + low;
if(i > high) i = high;
}
return(i);
}


float rndreal(lo ,hi)
/* real random number between specified limits */
float lo, hi;
{
```

```
return((randomperc() * (hi - lo)) + lo);
}


warmup_random(random_seed)
/* Get random off and running */
float random_seed;
{
int j1, ii;
double new_random, prev_random;

oldrand[54] = random_seed;
new_random = 0.000000001;
prev_random = random_seed;
for(j1 = 1 ; j1 <= 54; j1++)
{
ii = (21*j1)%54;
oldrand[ii] = new_random;
new_random = prev_random-new_random;
if(new_random<0.0) new_random = new_random + 1.0;
prev_random = oldrand[ii];
}

advance_random();
advance_random();
advance_random();

jrand = 0;
}
/*-------------------------------------------------------------
---------------*/
/* report.c - geeration report files
      */
/*-------------------------------------------------------------
---------------*/


report()
/* Write the population report */
{
void   repchar(), skip();
int m1,m2,l;

fprintf(outfp,"\nFitness=%d",f1);
```

```
/* application dependent report */
for(m1=0;m1<nd;m1++)
{
skip(outfp,2);
repchar(outfp,"*",20);
fprintf(outfp,"%15s",D1[m1]);
repchar(outfp,"*",25);
skip(outfp,2);
for(l=9;l<=12;l++)
{
repchar(outfp," ",10);
fprintf(outfp,"%2d-%2d",l,l+1);
}
skip(outfp,1);
repchar(outfp," ",2);
repchar(outfp,"-",65);
skip(outfp,1);

for(m2=0;m2<nr;m2++)
{
/* repchar(outfp," ",2);*/
fprintf(outfp,"%2s",R1[m2]);
for(l=0;l<ns;l++)
{
repchar(outfp,"|",1);
repchar(outfp," ",15);
}
repchar(outfp,"|",1);
skip(outfp,1);
repchar(outfp," ",2);

for(l=0;l<ns;l++)
{
repchar(outfp,"|",1);
repchar(outfp," ",1);
if(AA[m1][m2][l]<nc)
fprintf(outfp,"%14s",TT1[AA[m1][m2][l]]);
else
repchar(outfp," ",14);
/*   repchar(outfp," ",2); */
}
repchar(outfp,"|",1);
skip(outfp,1);
repchar(outfp," ",2);
for(l=0;l<ns;l++)
```

```
for(l=0;l<ns;l++)
{
int c;
c=0;
repchar(outfp,"|",1);
repchar(outfp," ",1);
if(AA[m1][m2][l]<nc)
{
c=PP[m1][m2][l];
fprintf(outfp,"%14d",c);
}
else
repchar(outfp," ",14);
}
repchar(outfp,"|",1);
skip(outfp,1);
repchar(outfp," ",2);
for(l=0;l<ns;l++)
{
repchar(outfp,"|",1);
repchar(outfp," ",15);
}
repchar(outfp,"|",1);
skip(outfp,1);
repchar(outfp," ",2);

repchar(outfp,"-",65);
skip(outfp,1);
}
}
}
```

```
/*-------------------------------------------------------------
---------------*/
/* utility.c - utility routines, contains copyright,  repchar,
skip           */
/*-------------------------------------------------------------
---------------*/
```

```
void repchar (outfp,ch,repcount)
/* Repeatedly write a character to stdout */
FILE *outfp;
char *ch;
int repcount;
{
int j;

for (j = 1; j <= repcount; j++) fprintf(outfp,"%s", ch);
}


void skip(outfp,skipcount)
/* Skip skipcount lines */
FILE *outfp;
int skipcount;
{
int j;

for (j = 1; j <= skipcount; j++) fprintf(outfp,"\n");
return;
}
```